Vector Semantics & Embeddings

Vector Semantics

Desiderata

Concepts or word senses

- Have a complex many-to-many association with **words** (homonymy, multiple senses)

Have relations with each other

- Synonymy
- Antonymy
- Similarity
- Relatedness
- Connotation

Defining meaning as a point in space based on distribution

Each word = a vector (not just "good" or "w_{45}")

Similar words are "**nearby in semantic space**"

We build this space by seeing which words are **nearby in text**

We'll discuss 2 kinds of embeddings

tf-idf

- Information Retrieval workhorse!
- A common baseline model
- **Sparse** vectors
- Words are represented by (a simple function of) the **counts** of nearby words

Word2vec

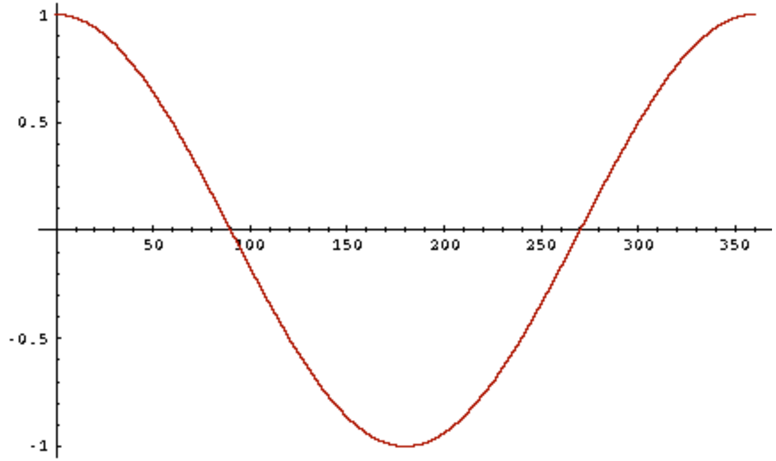
- **Dense** vectors
- Representation is created by training a classifier to **predict** whether a word is likely to appear nearby
- Later we'll discuss extensions called **contextual embeddings**

Cosine as a similarity metric

-1: vectors point in opposite directions

+1: vectors point in same directions

0: vectors are orthogonal



But since raw frequency values are non-negative, the cosine for term-term matrix vectors ranges from 0–1

Solution 1: tf-idf

tf-idf: Term Frequency - Inverse Document Frequency

Term Frequency:

$$\mathrm{tf}_{t,d} = \mathrm{count}(t,d)$$

$$\mathrm{tf}_{t,d} = \log_{10}(\mathrm{count}(t,d)+1)$$

Inverse Document Frequency:

$$\mathrm{idf}_t = \frac{N}{\mathrm{df}_t}$$

$$\mathrm{idf}_t = \log_{10}\left(\frac{N}{\mathrm{df}_t}\right)$$

tf-idf: $w_{t,d} = \mathrm{tf}_{t,d} \times \mathrm{idf}_t$

Vector Semantics & Embeddings

Word2vec

Sparse versus dense vectors

tf-idf (or PMI) vectors are

- **long** (length |V|= 20,000 to 50,000)
- **sparse** (most elements are zero)

Alternative: learn vectors which are

- **short** (length 50-1000)
- **dense** (most elements are non-zero)

Sparse versus dense vectors

Why dense vectors?

- Short vectors may be easier to use as **features** in machine learning (fewer weights to tune)
- Dense vectors may **generalize** better than explicit counts
- Dense vectors may do better at capturing synonymy:
  - *car* and *automobile* are synonyms; but are distinct dimensions
    - a word with *car* as a neighbor and a word with *automobile* as a neighbor should be similar, but aren't
- **In practice, they work better**

Common methods for getting short dense vectors

"Neural Language Model"-inspired models

- Word2vec (skipgram, CBOW), GloVe

Singular Value Decomposition (SVD)

- A special case of this is called LSA – Latent Semantic Analysis

Alternative to these "static embeddings":

- Contextual Embeddings (ELMo, BERT)
- Compute distinct embeddings for a word in its context
- Separate embeddings for each token of a word

Simple static embeddings you can download!

Word2vec (Mikolov et al)

https://code.google.com/archive/p/word2vec/

GloVe (Pennington, Socher, Manning)

http://nlp.stanford.edu/projects/glove/

Word2vec

Popular embedding method

Very fast to train

Code available on the web

Idea: **predict** rather than **count**

Word2vec provides various options. We'll discuss:

skip-gram with negative sampling (SGNS)

Word2vec

Instead of **counting** how often each word *w* occurs near "*apricot*"

- Train a classifier on a binary **prediction** task:
  - Is *w* likely to show up near "*apricot*"?

We don’t actually care about this task

- But we'll take the learned classifier weights as the word embeddings

Big idea: **self-supervision**:

- A word c that occurs near apricot in the corpus cats as the gold "correct answer" for supervised learning
- No need for human labels
- Bengio et al. (2003); Collobert et al. (2011)

Approach: predict if candidate word *c* is a "neighbor"

1. Treat the target word *t* and a neighboring context word *c* as **positive examples**.

Bag-of-words assumption

"put the sugar in the jam"

±3 words

Positive Example: (sugar, jam) as co-occuring

(sugar, the

(sugar, in)

(sugar, put)

2. Randomly sample other words in the vocabulary to create negative examples

3. Train a logistic regression classifier to distinguish positive & negative examples

Skip-Gram Training Data

Assume a +/- 2 word window, given training sentence:

…lemon, a [tablespoon of apricot jam, a] pinch…

c1 c2 [target] c3 c4

Skip-Gram Classifier

(assuming a +/- 2 word window)

...lemon, a [tablespoon of apricot jam, a] pinch...

c1 c2 [target] c3 c4

Goal: train a classifier that is given a candidate (**w**ord, **c**ontext) pair

Positive: (apricot, jam)

Negative: (apricot, aardvark)

...

And assigns each pair a probability:

$P(+ \mid w, c)$

$P(- \mid w, c) = 1 - P(+ \mid w, c)$

Similarity is computed from dot product

Remember: two vectors are similar if they have a high dot product

- Cosine is just a normalized dot product

So:

- Similarity(w,c) $\propto$ w · c

We'll need to normalize to get a probability

- (cosine isn't a probability either)

sigmoid : $\sigma(x) = \frac{1}{1+\exp(-x)}$

Turning dot products into probabilities

$\mathrm{Sim}(\mathrm{w},\mathrm{c}) \approx \mathrm{w} \cdot c$

To turn this into a probability, we'll use the *sigmoid function*:

Turning dot products into probabilities

$\text{Sim(w,c)} \approx \text{w} \cdot c$

To turn this into a probability, we'll use the *sigmoid function*:

$$P(+|w,c) = \sigma(c \cdot w) = \frac{1}{1+\exp(-c \cdot w)}$$

$$P(-|w,c) = 1 - P(+|w,c)$$

w = vector representation of target word
c = vector representation of context word

How Skip-Gram Classifier computes $P(+|w, c)$

$$P(+|w,c) = \sigma(\mathbf{c} \cdot \mathbf{w}) = \frac{1}{1+\exp(-\mathbf{c} \cdot \mathbf{w})}$$

This is for one context word, but we have lots of context words.
We'll assume independence and multiply them:

c = positive examples

$$P(+|w, c_{1:L}) = \prod_{i=1}^{L} \sigma(c_i \cdot w)$$

$$\log P(+|w, c_{1:L}) = \sum_{i=1}^{L} \log \sigma(c_i \cdot w)$$

Probability of w appearing in window $c_{1:L}$

Skip-gram classifier: summary

Goal 1: Maximize probability of w in observed context window

Goal 2: Minimize probability of w co-occuring w/ randomly sampled words

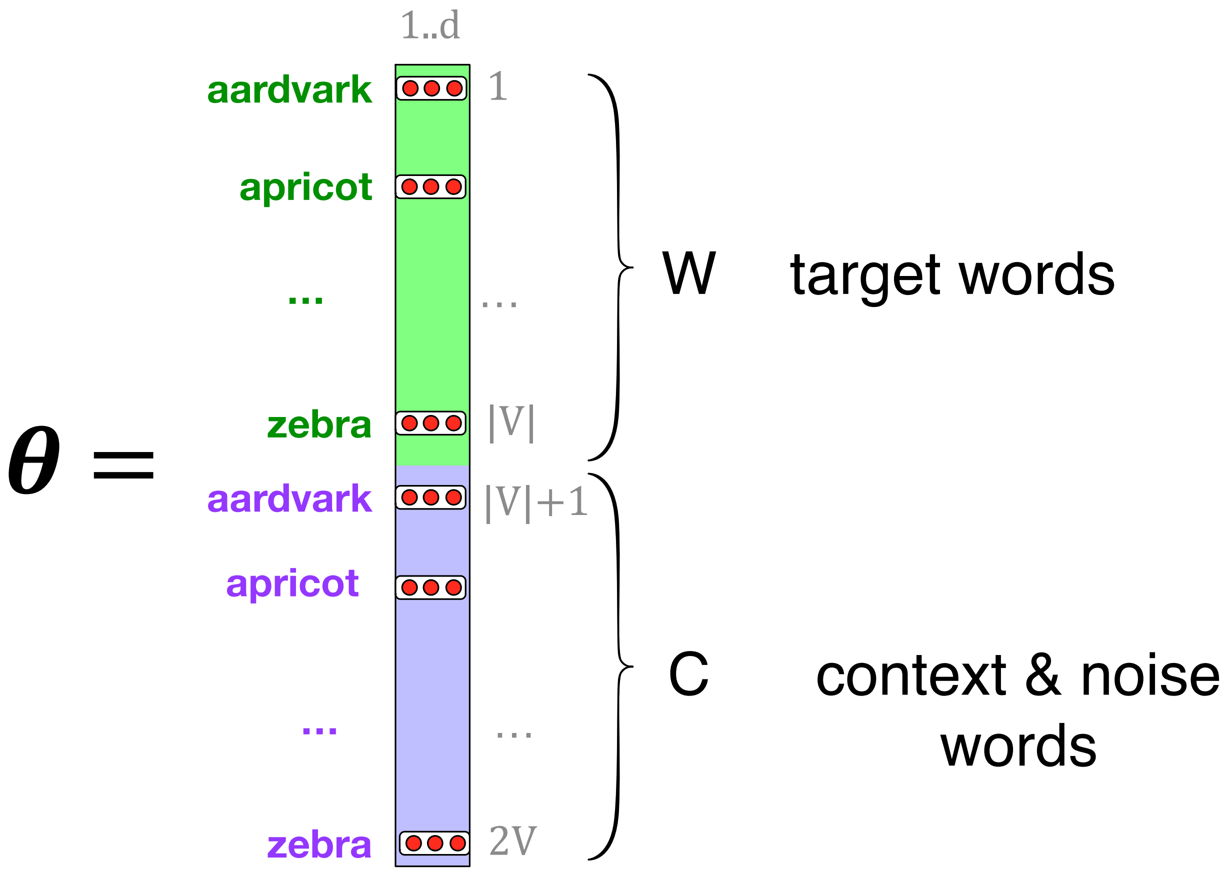
A probabilistic classifier, given

- a test target word *w*
- its context window of *L* words $c_{1:L}$

Estimates probability that w occurs in this window based on similarity of w (embeddings) to $c_{1:L}$ (embeddings).

To compute this, we just need embeddings for all the words.

These embeddings we'll need: a set for w, a set for c



Word2vec: Learning embeddings

Skip-Gram Training data

...lemon, a [tablespoon of apricot jam, a] pinch...

c1 c2 [target] c3 c4

Sampling Negatives: for each positive example, we'll grab k negative examples, sampling by frequency from our vocab.

positive examples +

w	c_{pos}
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

Negative examples −

W	Cneg
apricot	aadvark
	my
	where
	coaxial

Skip-Gram Training data

…lemon, a [tablespoon of apricot jam, a] pinch…

c1 c2 [target] c3 c4

positive examples +

w	c_{pos}
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

negative examples -

w	c_{neg}	w	c_{neg}
apricot	aardvark	apricot	seven
apricot	my	apricot	forever
apricot	where	apricot	dear
apricot	coaxial	apricot	if

Word2vec: how to learn vectors

Given the set of positive and negative training instances, and an initial set of embedding vectors

Goal: adjust the word vectors so they

- Maximize the similarity of the target word, context word positive pairs: (w, c_{pos})
- Minimize the similarity of the (w, c_{neg}) pairs

Loss function for one w *with* c_{pos} , c_{neg1} ...c_{negk}

Maximize the similarity of the target with the actual context words, and minimize the similarity of the target with the *k* negative sampled non-neighbor words.

$$p(+ \mid w, c_{pos}) \quad p(- \mid w, c_{neg})$$

$$L_{CE} = -\left[\log[\sigma(w \cdot c_{pos})] \quad \log[\sigma(\underline{-w \cdot c_{neg}})]\right]$$

More than 1 negative example:

$$= -\left[\log \sigma(c_{pos} \cdot w) + \sum^{k} \log \sigma(\underline{-c_{neg} \cdot w})\right]$$

for each individual word

Learning the classifier

How to learn?

- Stochastic gradient descent!

We'll adjust the word weights to

- make the positive pairs more likely
- and the negative pairs less likely,
- over the entire training set.

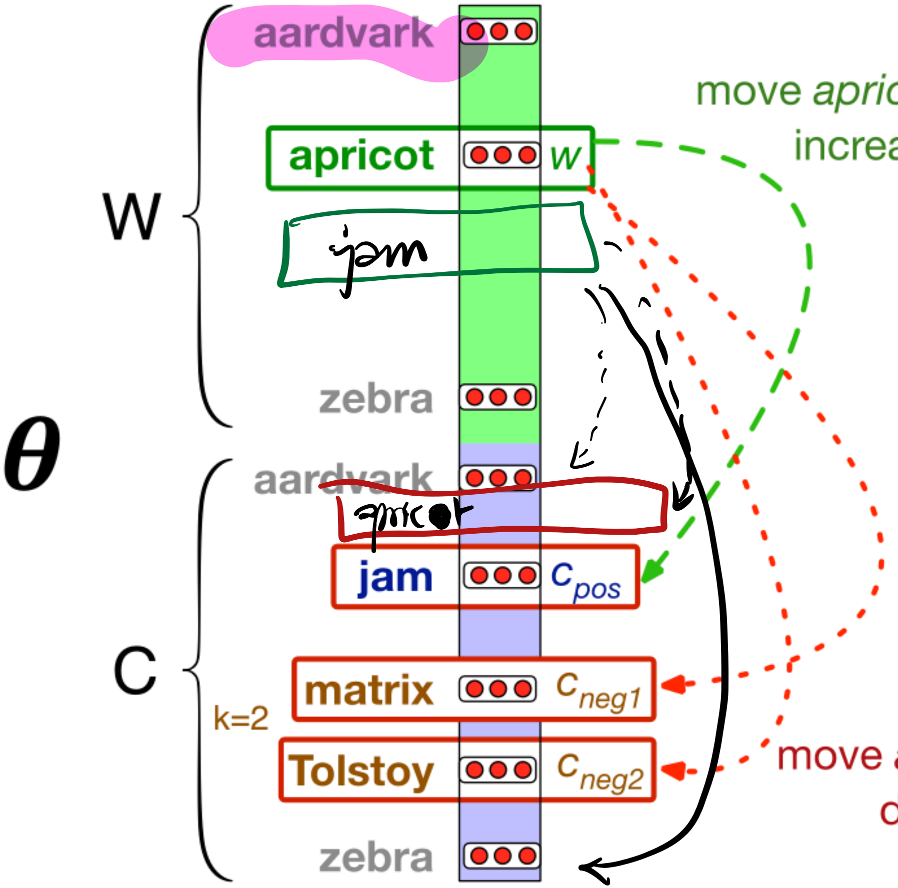
Intuition of one step of gradient descent

$c_{apricot}$: apricot + jam

$w_{apricot}$

move *apricot* and *jam* closer, increasing $c_{pos} \cdot w$

$w_{apricot}$: apricot + matrix



"…apricot jam…"

move *apricot* and *matrix* apart decreasing $c_{neg1} \cdot w$

move *apricot* and *Tolstoy* apart decreasing $c_{neg2} \cdot w$

Two sets of embeddings

SGNS learns two sets of embeddings

Target embeddings matrix W

Context embedding matrix C

It's common to just add them together, representing word *i* as the vector $w_i + c_i$

Summary: How to learn word2vec (skip-gram) embeddings

Start with V random d-dimensional vectors as initial embeddings

Train a classifier based on embedding similarity

- Take a corpus and take pairs of words that co-occur as positive examples
- Take pairs of words that don't co-occur as negative examples
- Train the classifier to distinguish these by slowly adjusting all the embeddings to improve the classifier performance
- Throw away the classifier code and keep the embeddings.

Vector Semantics & Embeddings

Properties of Embeddings

Gender in NLP

Harms of Gender Exclusivity and Challenges in Non-Binary Representation in Language Technologies

Sunipa Dev
she/her
UCLA

Masoud Monajatipoor*
he/him
UCLA

Anaelia Ovalle*
they/he/she
UCLA

Arjun Subramonian*
they/them
UCLA, Queer in AI

Jeff M Phillips
he/him
University of Utah

Kai-Wei Chang
he/him
UCLA

Abstract

Content Warning: This paper contains examples of stereotypes and associations, misgendering, erasure, and other harms that could be offensive and triggering to trans and non-binary individuals.

Gender is widely discussed in the context of language tasks and when examining the stereo-

A bulk of social bias studies on language models have focused on binary gender and the stereotypes associated with masculine and feminine attributes (Bolukbasi et al., 2016; Webster et al., 2018; Dev et al., 2020b). Additionally, models often rely on gendered information for decision making, such as in named entity recognition, coreference resolution, and machine translation (Mehrabi

The kinds of neighbors depend on window size

Small windows (C= +/- 2) : nearest words are syntactically similar words in same taxonomy

- *Hogwarts* nearest neighbors are other fictional schools
  - *Sunnydale, Evernight, Blandings*

Large windows (C= +/- 5) : nearest words are related words in same semantic field

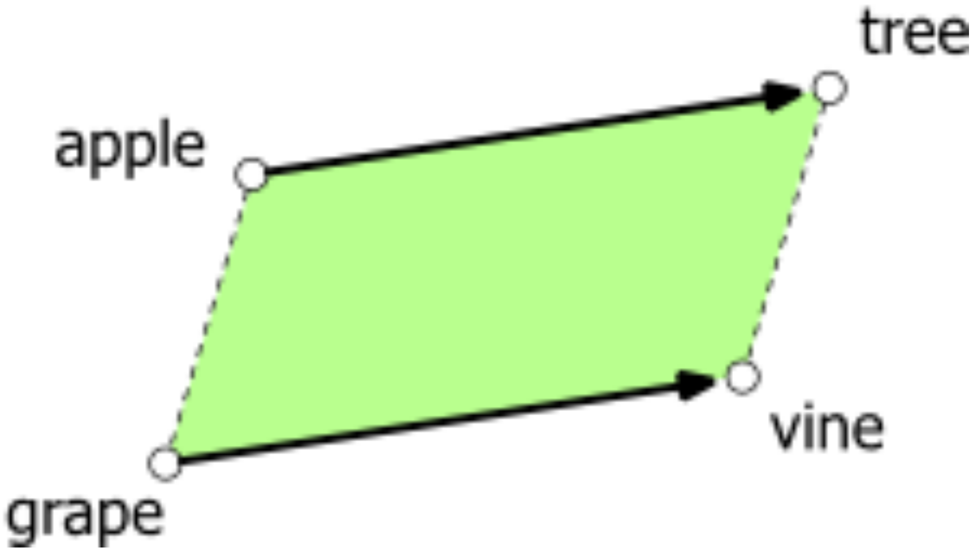
- *Hogwarts* nearest neighbors are Harry Potter world:
  - *Dumbledore, half-blood, Malfoy*

Analogical relations

The classic parallelogram model of analogical reasoning (Rumelhart and Abrahamson 1973)

To solve: "*apple is to tree as grape is to* _____"

Add $\overrightarrow{\textit{tree}} - \overrightarrow{\textit{apple}}$ *to* $\overrightarrow{\textit{grape}}$ *to get* $\overrightarrow{\textit{vine}}$



Analogical relations via parallelogram

The parallelogram method can solve analogies with both sparse and dense embeddings (Turney and Littman 2005, Mikolov et al. 2013b)

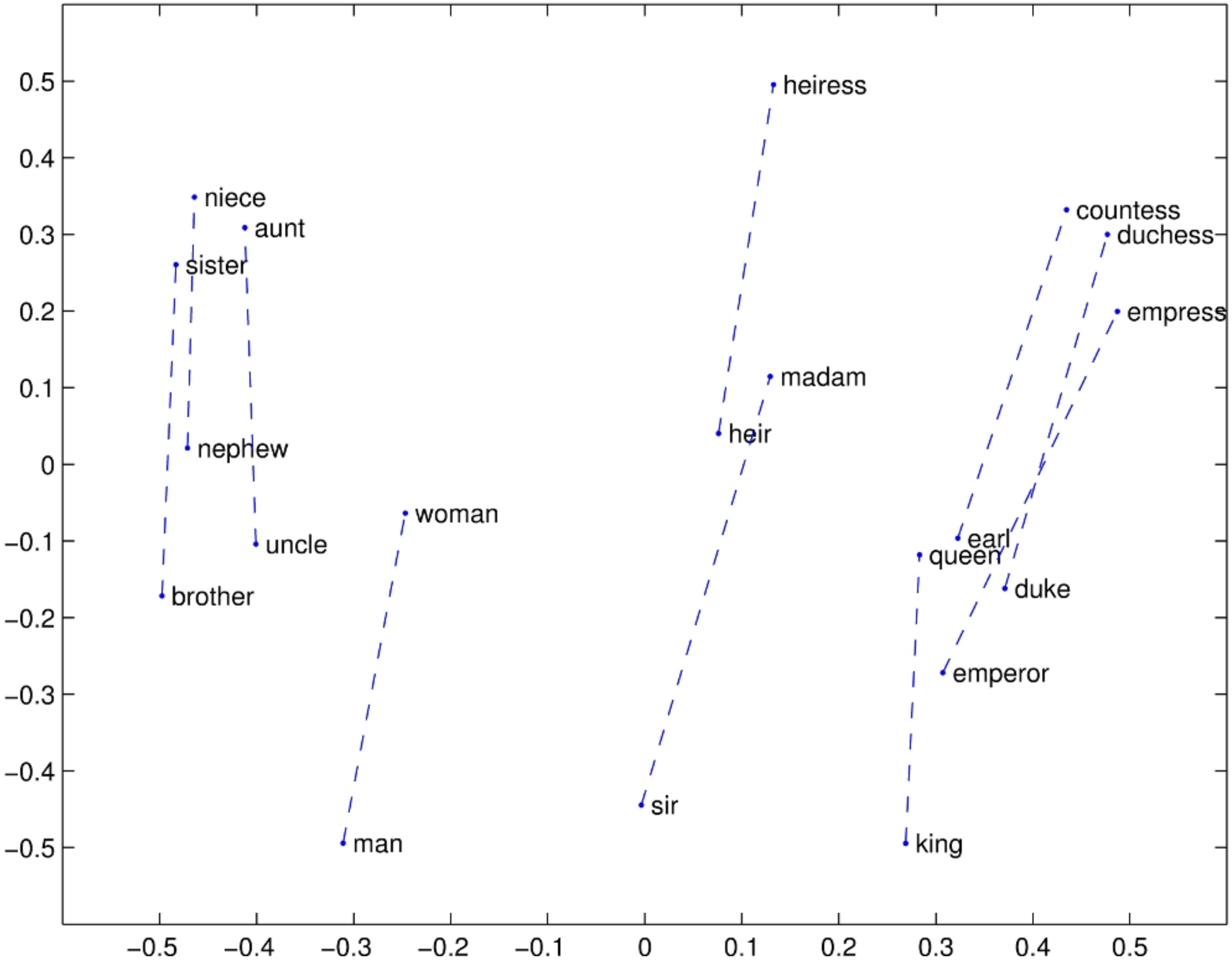
$\overrightarrow{\text{king}} - \overrightarrow{\text{man}} + \overrightarrow{\text{woman}}$ is close to $\overrightarrow{\text{queen}}$

$\overrightarrow{\text{Paris}} - \overrightarrow{\text{France}} + \overrightarrow{\text{Italy}}$ is close to $\overrightarrow{\text{Rome}}$

For a problem a:a*::b:b*, the parallelogram method is:

$$\hat{\mathbf{b}}^* = \underset{\mathbf{x}}{\operatorname{argmin}} \ \text{distance}(\mathbf{x}, \mathbf{b} - \mathbf{a} + \mathbf{a}^*)$$

Structure in GloVE Embedding space



Caveats with the parallelogram method

It only seems to work for frequent words, small distances and certain relations (relating countries to capitals, or parts of speech), but not others. (Linzen 2016, Gladkova et al. 2016, Ethayarajh et al. 2019a)

Understanding analogy is an open area of research (Peterson et al. 2020)

Embeddings as a window onto historical semantics

Train embeddings on different decades of historical text to see meanings shift

~30 million books, 1850-1990, Google Books data

William L. Hamilton, Jure Leskovec, and Dan Jurafsky. 2016. Diachronic Word Embeddings Reveal Statistical Laws of Semantic Change. Proceedings of ACL.

Embeddings reflect cultural bias!

Bolukbasi, Tolga, Kai-Wei Chang, James Y. Zou, Venkatesh Saligrama, and Adam T. Kalai. "Man is to computer programmer as woman is to homemaker? debiasing word embeddings." In *NeurIPS*, pp. 4349-4357. 2016.

Ask "Paris : France :: Tokyo : x"

- x = Japan

Ask "father : doctor :: mother : x"

- x = nurse

Ask "man : computer programmer :: woman : x"

- x = homemaker

Algorithms that use embeddings as part of e.g., hiring searches for programmers, might lead to bias in hiring

Historical embedding as a tool to study cultural biases

Garg, N., Schiebinger, L., Jurafsky, D., and Zou, J. (2018). Word embeddings quantify 100 years of gender and ethnic stereotypes. Proceedings of the National Academy of Sciences 115(16), E3635–E3644.

- Compute a **gender or ethnic bias** for each adjective: e.g., how much closer the adjective is to "woman" synonyms than "man" synonyms, or names of particular ethnicities
  - Embeddings for **competence** adjective (*smart, wise, brilliant, resourceful, thoughtful, logical)* are biased toward men, a bias slowly decreasing 1960-1990
  - Embeddings for **dehumanizing** adjectives (barbaric, monstrous, bizarre) were biased toward Asians in the 1930s, bias decreasing over the 20th century.
- These match the results of old surveys done in the 1930s