



CMPT 413/713: Natural Language Processing

# Word Embeddings - Word2Vec

Spring 2025  
2025-01-27

Adapted from slides from Dan Jurafsky, Chris Manning, Danqi Chen and Karthik Narasimhan

# Representing words by their context

**Distributional hypothesis:** words that occur in similar contexts tend to have similar meanings



J.R.Firth 1957

- “You shall know a word by the company it keeps”
- One of the most successful ideas of modern statistical NLP!

*...government debt problems turning into **banking** crises as happened in 2009...*

*...saying that Europe needs unified **banking** regulation to replace the hodgepodge...*

*...India has just given its **banking** system a shot in the arm...*

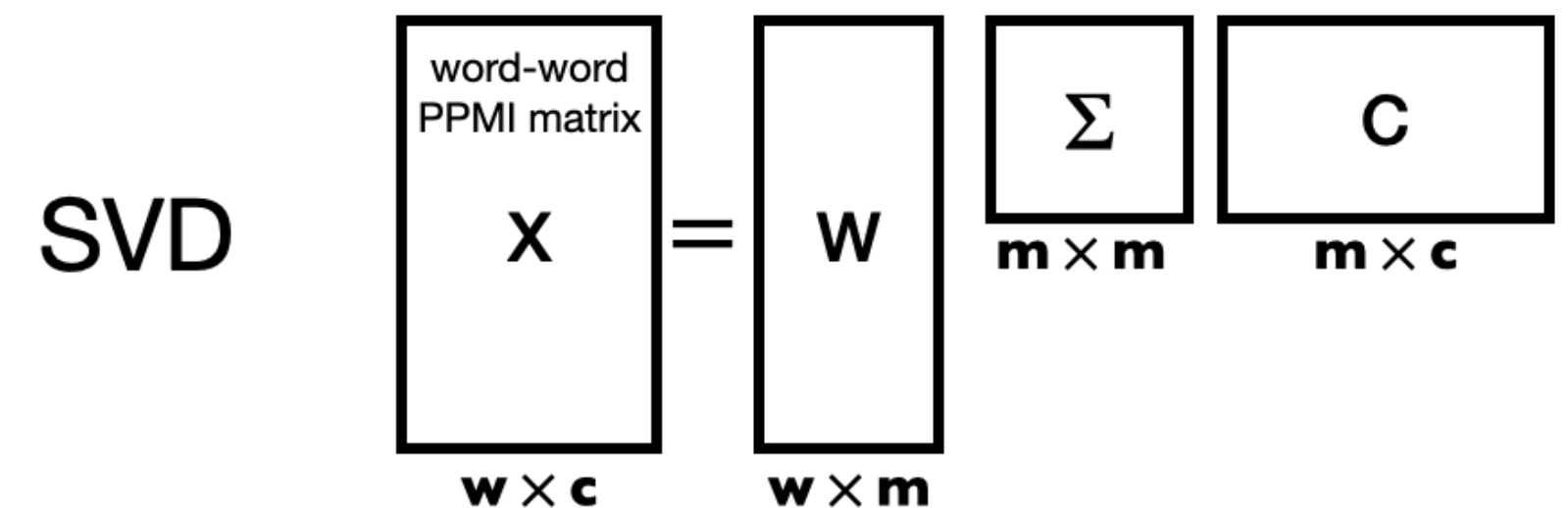
These **context words** will represent **banking**.

# Want to have dense vectors

- Short vectors are easier to use as features in ML systems
- Dense vectors may generalize better than storing explicit counts
- They do better at capturing synonymy
  - $w_1$  co-occurs with “car”,  $w_2$  co-occurs with “automobile”

- Different methods for getting dense vectors:
  - Singular value decomposition (SVD)
  - word2vec and friends: “learn” the vectors!

SVD


$$\begin{array}{|c|} \hline \text{word-word} \\ \text{PPMI matrix} \\ \hline \mathbf{X} \\ \hline \mathbf{w} \times \mathbf{c} \\ \hline \end{array} = \begin{array}{|c|} \hline \mathbf{W} \\ \hline \mathbf{w} \times \mathbf{m} \\ \hline \end{array} \begin{array}{|c|} \hline \mathbf{\Sigma} \\ \hline \mathbf{m} \times \mathbf{m} \\ \hline \end{array} \begin{array}{|c|} \hline \mathbf{C} \\ \hline \mathbf{m} \times \mathbf{c} \\ \hline \end{array}$$

Count based method  
(known since the 1990s)

# How are these embeddings learned?

Get embeddings by **counting** or by **predicting** (i.e. training a classifier)!

C1: A bottle of \_\_\_\_ is on the table.

C2: Everybody likes \_\_\_\_.

C3: Don't have \_\_\_\_ before you drive.

C4: We make \_\_\_\_ out of corn.

|           | C1 | C2 | C3 | C4 |
|-----------|----|----|----|----|
| tejuino   | 1  | 1  | 1  | 1  |
| loud      | 0  | 0  | 0  | 0  |
| motor-oil | 1  | 0  | 0  | 0  |
| tortillas | 0  | 1  | 0  | 1  |
| choices   | 0  | 1  | 0  | 0  |
| wine      | 1  | 1  | 1  | 0  |

Use as context: other words that appear in a span around the target word  
“words that occur in similar **contexts** tend to have similar meanings”

# How are these embeddings learned?

Learn predictor to fill in the blank!

C1: A bottle of \_\_\_\_ is on the table.

- Represent each word as a vector
- Train classifier to predict word using context words.
- During training, the word vector is updated so that it is possible to predict the center word using the context words

|           | bottle | likes | before | make | corn |
|-----------|--------|-------|--------|------|------|
| tejuino   | 1      | 1     | 1      | 1    | 1    |
| loud      | 0      | 0     | 0      | 0    | 0    |
| motor-oil | 1      | 0     | 0      | 0    | 0    |
| tortillas | 0      | 1     | 0      | 1    | 1    |
| choices   | 0      | 1     | 0      | 0    | 0    |
| wine      | 1      | 1     | 1      | 0    | 0    |

Use as context: other words that appear in a span around the target word  
“words that occur in similar contexts tend to have similar meanings”

# Many different ways to learn the representations

- From context words, predict target word (Masked LM)

A bottle of \_\_\_\_ is on the table.

- From target word, predict other context words.

What words go with “tejuino”?

- From previous words, predict next (target) word (Traditional LM)

A bottle of \_\_\_\_

A bottle of tejuino is on the \_\_\_\_

# Many different ways to learn the representations

- Different architectures and models to learn the representations
- Get sentence embeddings by combining word embeddings
- Two types of word embeddings
  - Static - mapping of word to embedding
  - Contextual - embedding of word changes based on the context

# Word2vec and friends

# Download pretrained word embeddings

Google

**Word2vec** (Mikolov et al.)

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

facebook

**Fasttext** <http://www.fasttext.cc/>



**Glove** (Pennington, Socher, Manning)

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



# Word2Vec

- Popular embedding method
- Very fast to train
- Idea: **predict** rather than **count**



Original word2vec formulation: Efficient Estimation of Word Representations in Vector Space (Mikolov et al, 2013a)

Negative sampling: Distributed Representations of Words and Phrases and their Compositionality (Mikolov et al, 2013b)

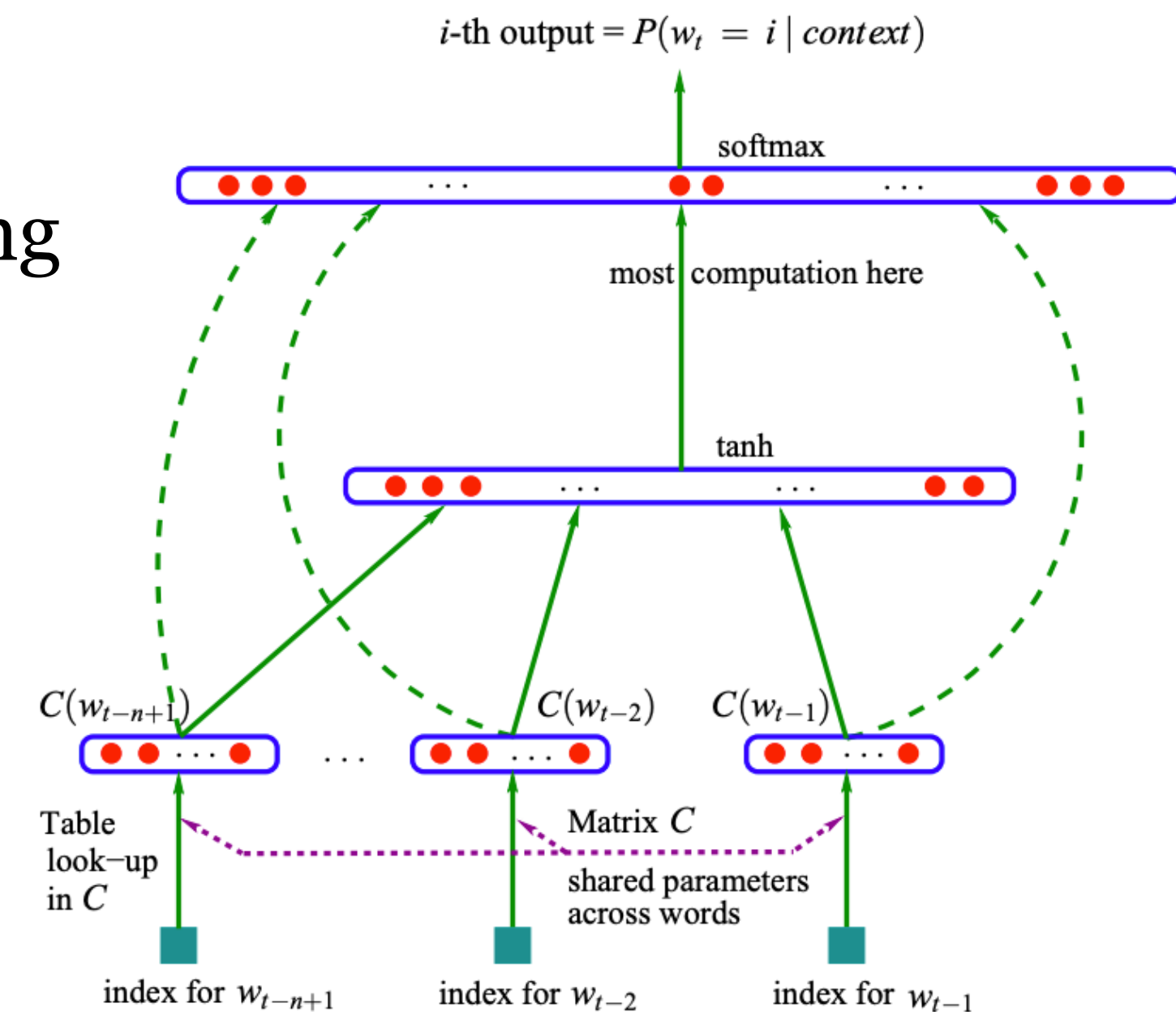
# Word2Vec

- Instead of counting how often each word  $w$  occurs near “apricot”
- Train a classifier on a **prediction** task:
  - Is  $w$  likely to show up near “apricot”?
  - There are different ways to formulate this prediction task
  - We will look at “Skip-gram with negative sampling”
- We don’t actually care about this task (pretext task)
  - But we’ll take the learned classifier **weights** as **the word embeddings**

# Word2Vec

Insight: use running text as implicitly supervised training data!

- A word  $s$  near apricot
  - Act as gold “correct answer” to the question “Is word  $w$  likely to show up near apricot?”
- No need for hand-labeled supervision
- The idea comes from neural language modeling
  - Bengio et al (2003)
  - Collobert et al (2011)



(Bengio et al, 2003): A Neural Probabilistic Language Model

# Word2vec

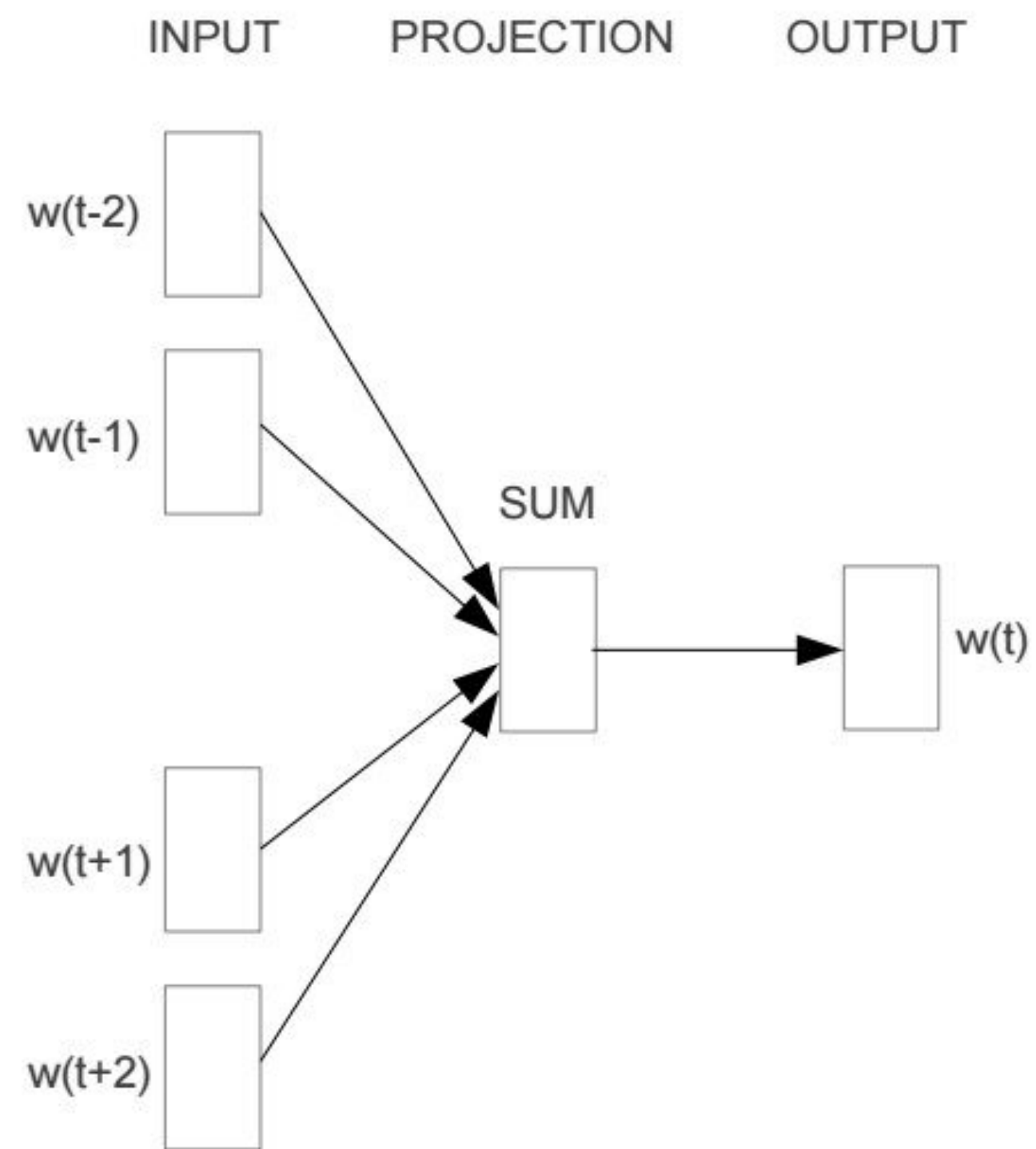
- Input: a large text corpora  $V, d$ 
  - $V$ : a pre-defined vocabulary
  - $d$ : dimension of word vectors (e.g. 300)
  - Text corpora (words  $w_1, \dots, w_T$ )
    - Wikipedia + Gigaword 5: 6B
    - Twitter: 27B
    - Common Crawl: 840B
- Output:  $f : V \rightarrow \mathbb{R}^d$

$$v_{\text{cat}} = \begin{pmatrix} -0.224 \\ 0.130 \\ -0.290 \\ 0.276 \end{pmatrix} \quad v_{\text{dog}} = \begin{pmatrix} -0.124 \\ 0.430 \\ -0.200 \\ 0.329 \end{pmatrix}$$
$$v_{\text{the}} = \begin{pmatrix} 0.234 \\ 0.266 \\ 0.239 \\ -0.199 \end{pmatrix} \quad v_{\text{language}} = \begin{pmatrix} 0.290 \\ -0.441 \\ 0.762 \\ 0.982 \end{pmatrix}$$

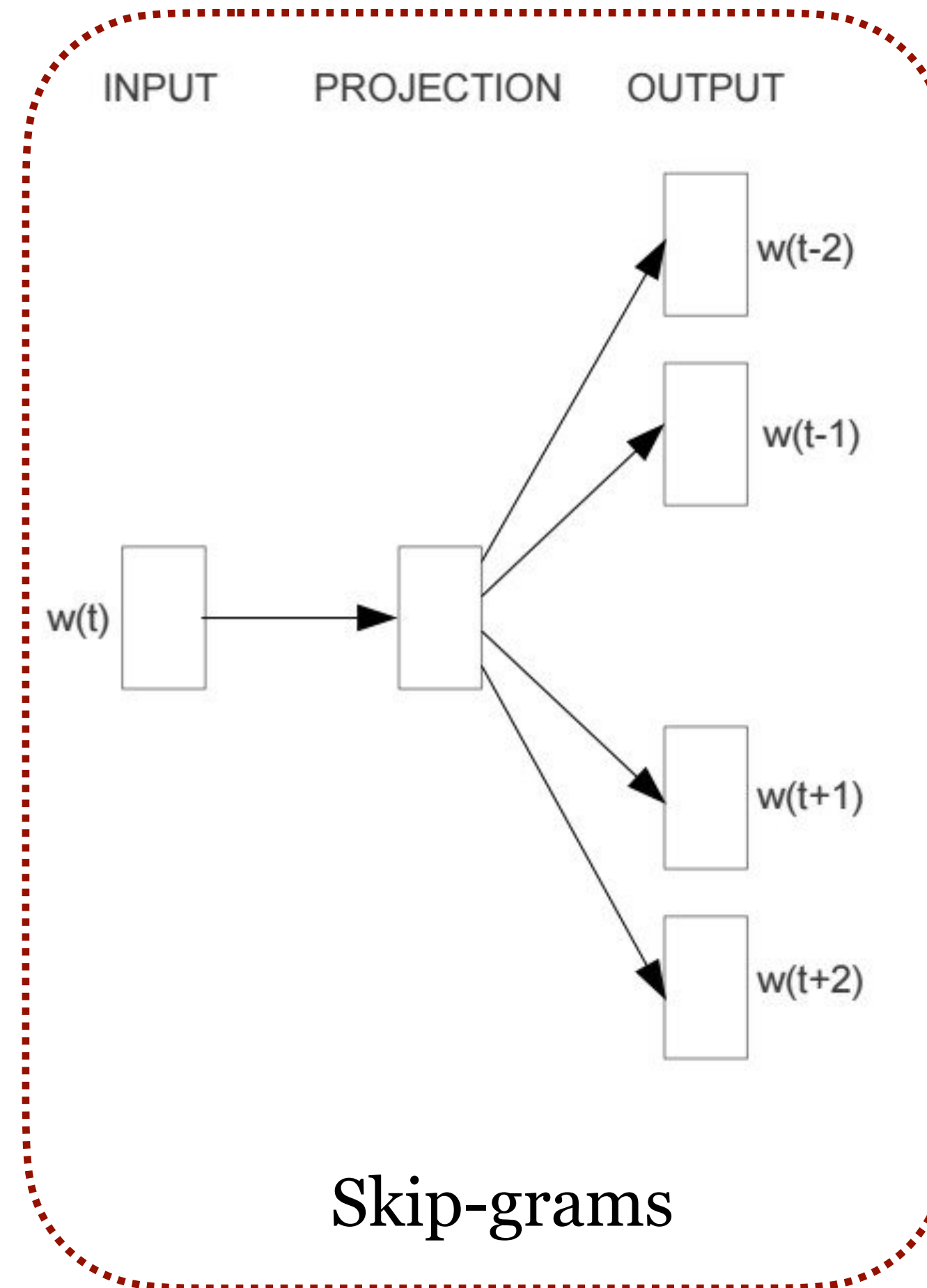
Learn  $f$  by training classifiers to **predict** words and take learned weights as word vectors.

# Word2vec

Predict **center** word from context words    Predict **context** words from center word



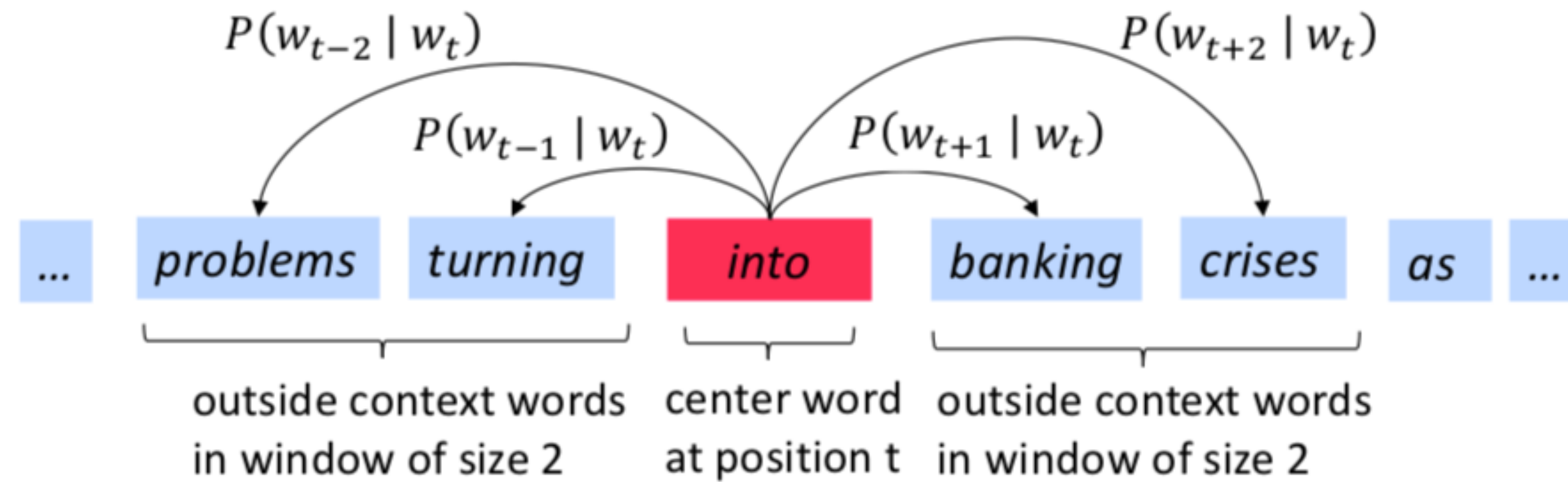
Continuous Bag of Words (CBOW)



Skip-grams

# Skip-gram

- The idea: we want to use words to **predict** their context words
- Context: a fixed window of size  $2m$  ( $m = 2$  in this example)



# Skip-gram: Basic Setup

Let's represent words as vectors of some length (say 300), randomly initialized.

So we start with  $300 * V$  random parameters

Over the entire training set, we'd like to adjust those word vectors such that we

- Predict the probability distribution for how likely a word  $c$  is to be a context word of a target word  $t$

$$P(c | t)$$

# Skip-gram

Training sentence:

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

c1 c2 t c3 c4

Training data

Training data: input/output pairs centering on apricot  
assume a +/- 2 word window

Given a tuple  $(t, c)$  = target, context

(apricot, jam)

(apricot, aardvark)

Goal

Return probability that  $c$  is a real context word:

$$P(c | t)$$

Probability distribution  
over vocabulary

# Skip-gram: objective function

- For each position  $t = 1, 2, \dots, T$ , predict context words within context size  $m$ , given center word  $w_t$ :

$$\mathcal{L}(\theta) = \prod_{t=1}^T \prod_{-m \leq j \leq m, j \neq 0} P(w_{t+j} \mid w_t; \theta)$$

all the parameters to be optimized

Subscript refer to position in corpus  
 $w_{t+j}$  refer to word in vocabulary  $V$

- The objective function  $J(\theta)$  is the (average) negative log likelihood:

$$J(\theta) = -\frac{1}{T} \log \mathcal{L}(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{-m \leq j \leq m, j \neq 0} \log P(w_{t+j} \mid w_t; \theta)$$

# How to define $P(w_{t+j} \mid w_t; \theta)$ ?

- We have two sets of vectors for each word in the vocabulary

$\mathbf{u}_t \in \mathbb{R}^d$  : embedding for target word  $t$

$\mathbf{v}_c \in \mathbb{R}^d$  : embedding for context word  $c$

- Use inner product  $\mathbf{u}_t \cdot \mathbf{v}_c$  to measure how likely word  $t$  appears with context word  $c$ , the larger the better

$$P(w_{t+j} \mid w_t) = \frac{\exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_{w_{t+j}})}{\sum_{k \in V} \exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_k)}$$

“softmax” we learned before!

← Normalized over entire vocabulary

$\theta = \{\{\mathbf{u}_k\}, \{\mathbf{v}_k\}\}$  are all the parameters in this model!

Q: Why two sets of vectors?

# How to train the model

Objective function: average log likelihood

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{-m \leq j \leq m, j \neq 0} \log P(w_{t+j} \mid w_t; \theta)$$

Parameters:

**Q: How many parameters are in total?**

$$\theta = \{\{\mathbf{u}_k\}, \{\mathbf{v}_k\}\}$$

Note: because we need to learn both set of parameters, this is a non-convex objective function

We can apply stochastic gradient descent (SGD)!

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_{\theta} J(\theta)$$

Need to compute:  $\nabla_{\theta} J(\theta) = ?$

# Gradients for word2vec

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{-m \leq j \leq m, j \neq 0} \log \frac{\exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_{w_{t+j}})}{\sum_{k \in V} \exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_k)}$$

- Consider one pair of center / context words  $(w_t, w_c)$ ,  $w_t, w_c \in V$
- We need to compute the gradient of  $y$  with respect to  $\mathbf{u}_t$  and  $\mathbf{v}_k$ ,  $\forall k \in V$

$$y = -\log \left( \frac{\exp(\mathbf{u}_t \cdot \mathbf{v}_c)}{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)} \right)$$

# Computing the gradients

Consider one pair of target/context words (t, c):

$$y = -\log \left( \frac{\exp(\mathbf{u}_t \cdot \mathbf{v}_c)}{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)} \right) = -\mathbf{u}_t \cdot \mathbf{v}_c + \log \left( \sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k) \right) \quad \forall k \in V$$

$$\begin{aligned} \frac{\partial y}{\partial \mathbf{u}_t} &= \frac{\partial(-\mathbf{u}_t \cdot \mathbf{v}_c + \log(\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)))}{\partial \mathbf{u}_t} \\ &= -\mathbf{v}_c + \frac{\sum_{k \in V} \frac{\partial \exp(\mathbf{u}_t \cdot \mathbf{v}_k)}{\partial \mathbf{u}_t}}{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)} \\ &= -\mathbf{v}_c + \frac{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k) \mathbf{v}_k}{\sum_{k' \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_{k'})} \\ &= -\mathbf{v}_c + \sum_{k \in V} \frac{\exp(\mathbf{u}_t \cdot \mathbf{v}_k)}{\sum_{k' \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_{k'})} \mathbf{v}_k \\ &= -\mathbf{v}_c + \sum_{k \in V} P(k|t) \mathbf{v}_k \end{aligned}$$

$$\begin{aligned} \frac{\partial y}{\partial \mathbf{v}_k} &= \frac{\partial(-\mathbf{u}_t \cdot \mathbf{v}_c + \log(\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)))}{\partial \mathbf{v}_k} \\ &= -1[k = c] \mathbf{u}_t + P(k|t) \mathbf{u}_t \end{aligned}$$

We can pull  $\sum_{k \in V}$  out and push the denominator inside the sum

Note we defined  $P(w_{t+j} \mid w_t) = \frac{\exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_{w_{t+j}})}{\sum_{k \in V} \exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_k)}$

Make sure you know how to do this!

# Putting it together

- Input: text corpus, context size  $m$ , embedding size  $d$ ,  $V$
- Initialize  $\mathbf{u}_t, \mathbf{v}_k$  randomly
- Walk through the training corpus and collect training data (t, c):
  - Update  $\mathbf{u}_t \leftarrow \mathbf{u}_t - \eta \frac{\partial y}{\partial \mathbf{u}_t}$
  - Update  $\mathbf{v}_k \leftarrow \mathbf{v}_k - \eta \frac{\partial y}{\partial \mathbf{v}_k}, \forall k \in V$

Any issues?

# Problem with Naive softmax

Problem: every time you get one pair of  $(t, c)$ , you need to update  $\mathbf{u}_t$  for each  $\mathbf{v}_k$ , and update  $\mathbf{v}_k$  for all the words in the vocabulary! It is very computationally expensive.

$$\frac{\partial y}{\partial \mathbf{u}_t} = -\mathbf{v}_c + \sum_{k \in V} P(k|t) \mathbf{v}_k$$

$$\frac{\partial y}{\partial \mathbf{v}_k} = -1[k = c] \mathbf{u}_t + P(k|t) \mathbf{u}_t \quad \forall k \in V$$

**Negative sampling:** instead of considering all the words in  $V$ , let's randomly sample  $K$  (5-20) negative examples.

softmax: 
$$y = -\log \left( \frac{\exp(\mathbf{u}_t \cdot \mathbf{v}_c)}{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)} \right)$$

NS: 
$$y = -\log(\sigma(\mathbf{u}_t \cdot \mathbf{v}_c)) - \sum_{i=1}^K E_{j \sim P(w)} \log(\sigma(-\mathbf{u}_t \cdot \mathbf{v}_j))$$

# Skip-gram with negative sampling (SGNS)

Key idea: convert the  $|V|$ -way classification problem into a binary classification problem

$$y = -\log(\sigma(\mathbf{u}_t \cdot \mathbf{v}_c)) - \sum_{i=1}^K E_{j \sim P(w)} \log(\sigma(-\mathbf{u}_t \cdot \mathbf{v}_j))$$

Every time we get a pair of  $(t,c)$  words, predict if they co-occur together or not.  
(don't try to predict the correct  $c$  from amongst all the words in the vocabulary)

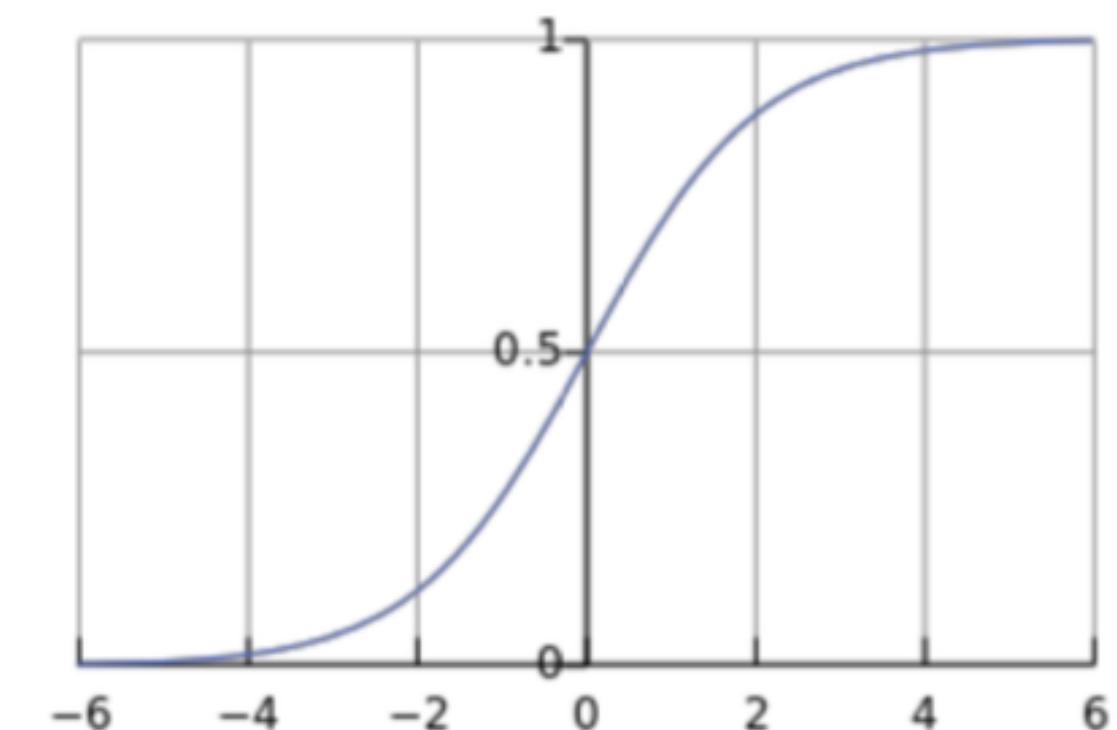
**positive examples +**

| t       | c          |
|---------|------------|
| apricot | tablespoon |
| apricot | of         |
| apricot | jam        |
| apricot | a          |

**negative examples -**

| t       | c        | t       | c       |
|---------|----------|---------|---------|
| apricot | aardvark | apricot | seven   |
| apricot | my       | apricot | forever |
| apricot | where    | apricot | dear    |
| apricot | coaxial  | apricot | if      |

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



Similar to training a **logistic regression** for binary classification (but need to optimize  $u$  and  $v$  together)

$$P(y = 1 \mid t, c) = \sigma(\mathbf{u}_t \cdot \mathbf{v}_c)$$

# Skip-gram with negative sampling (SGNS)

Basic formula:

1. Treat the target word and a neighboring context word as positive examples.
2. Randomly sample other words in the lexicon to get negative samples
3. Train a classifier to distinguish those two cases
4. Use the weights as the embeddings

# SGNS: Training the classifier

## Iterative process

Let's represent words as vectors of some length (say 300), randomly initialized.

So we start with  $300 * V$  random parameters

Over the entire training set, adjust those word vectors

- Maximize the similarity of the **target word**, **context word** pairs (t,c) drawn from the positive data
- Minimize the similarity of the (t,c) pairs drawn from the negative data.

# Skip-gram with negative sampling

Training sentence:

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

c1 c2 t c3 c4

Training data

Training data: input/output pairs centering on apricot  
assume a +/- 2 word window

Given a tuple  $(t, c)$  = target, context

(apricot, jam)

(apricot, aardvark)

Goal

Return probability that  $c$  is a real context word:

$$P(+ | t, c)$$

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

# SGNS Training

Training sentence:

... lemon, a tablespoon of apricot jam a pinch ...  
                  c1          c2  t      c3  c4

**positive examples +**

| t       | c          |
|---------|------------|
| apricot | tablespoon |
| apricot | of         |
| apricot | preserves  |
| apricot | or         |

- For each positive example, we'll create  $K$  negative examples.
- Using noise words
- Any random word that isn't  $t$

# Choosing noise words

Could pick  $w$  according to their unigram frequency  $P(w)$

More common to chose them according to  $P_\alpha(w)$

$$P_\alpha(w) = \frac{\text{count}(w)^\alpha}{\sum_w \text{count}(w)^\alpha}$$

$\alpha = 3/4$  works well because it gives rare noise words slightly higher probability

To show this, imagine two events  $p(a) = 0.99$  and  $p(b) = 0.01$ :

$$P_\alpha(a) = \frac{.99^{.75}}{.99^{.75} + .01^{.75}} = .97$$

$$P_\alpha(b) = \frac{.01^{.75}}{.99^{.75} + .01^{.75}} = .03$$

# SGNS: objective function

We want to maximize...

$$\sum_{(t,c) \in +} \log P(+|t, c) + \sum_{(t,c) \in -} \log P(-|t, c)$$

Maximize the + label for the pairs from the **positive** training data,  
and the – label for the pairs sample from the **negative** data.

Focusing on one target word  $t$ :

$$\begin{aligned} L(\boldsymbol{\theta}) &= \log P(+|t, c) + \sum_{i=1} \log P(-|t, n_i) \\ &= \log \sigma(c \cdot t) + \sum_{i=1}^k \log \sigma(-n_i \cdot t) \\ &= \log \frac{1}{1 + e^{-c \cdot t}} + \sum_{i=1}^k \log \frac{1}{1 + e^{n_i \cdot t}} \end{aligned}$$

# Skip-gram with negative sampling (SGNS)

Key idea: convert the  $|V|$ -way classification problem into a binary classification problem

$$y = -\log(\sigma(\mathbf{u}_t \cdot \mathbf{v}_c)) - \sum_{i=1}^K E_{j \sim P(w)} \log(\sigma(-\mathbf{u}_t \cdot \mathbf{v}_j))$$

Every time we get a pair of  $(t, c)$  words, predict if they co-occur together or not.  
(don't try to predict the correct  $c$  from amongst all the words in the vocabulary)

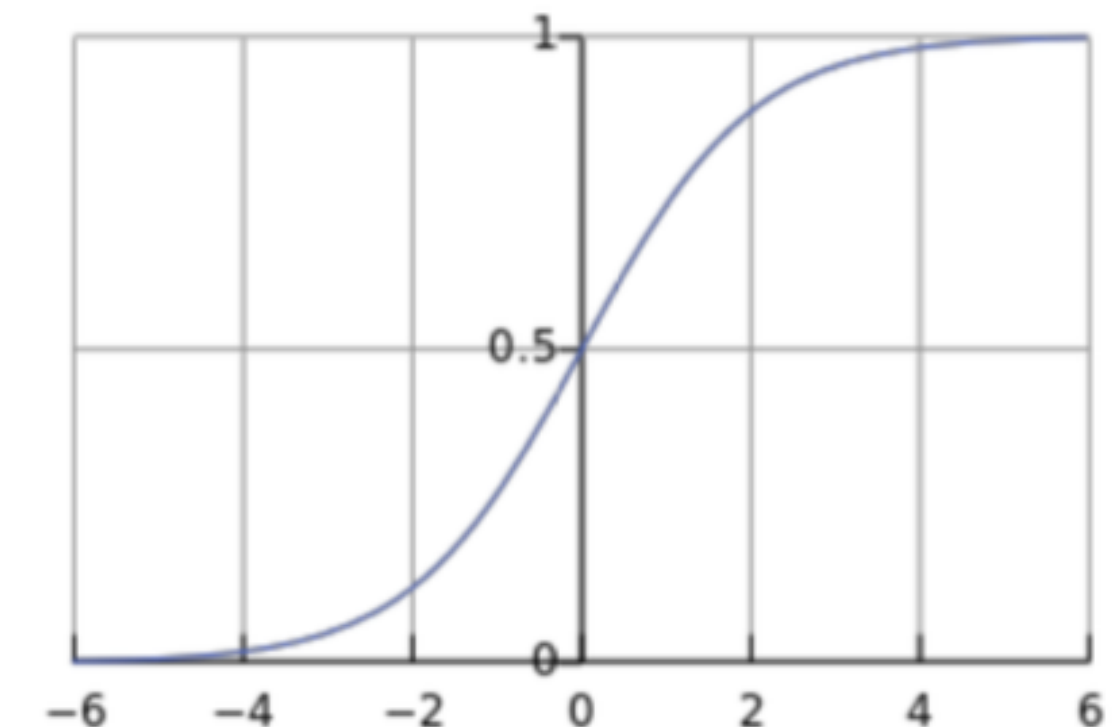
**positive examples +**

| t       | c          |
|---------|------------|
| apricot | tablespoon |
| apricot | of         |
| apricot | jam        |
| apricot | a          |

**negative examples -**

| t       | c        | t       | c       |
|---------|----------|---------|---------|
| apricot | aardvark | apricot | seven   |
| apricot | my       | apricot | forever |
| apricot | where    | apricot | dear    |
| apricot | coaxial  | apricot | if      |

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



Similar to training a **logistic regression** for binary classification (but need to optimize  $\mathbf{u}$  and  $\mathbf{v}$  together)

$$P(y = 1 \mid t, c) = \sigma(\mathbf{u}_t \cdot \mathbf{v}_c)$$

<http://jalammar.github.io/illustrated-word2vec/>

# Relationship of PMI and Word2Vec

- Word2Vec Skipgram with negative sampling (SGNS) implicitly factorizes word-context PMI matrix  $\mathbf{M}^{\text{PMI}}$

$$\mathbf{W} \cdot \mathbf{C}^T \approx \mathbf{M}^{\text{PMI}} - \log k$$

$$\vec{w} \cdot \vec{c} \approx \text{PMI}(w, c) - \log k$$

$\mathbf{W}$  is the matrix of word embeddings

$\mathbf{C}$  is the matrix of context embeddings

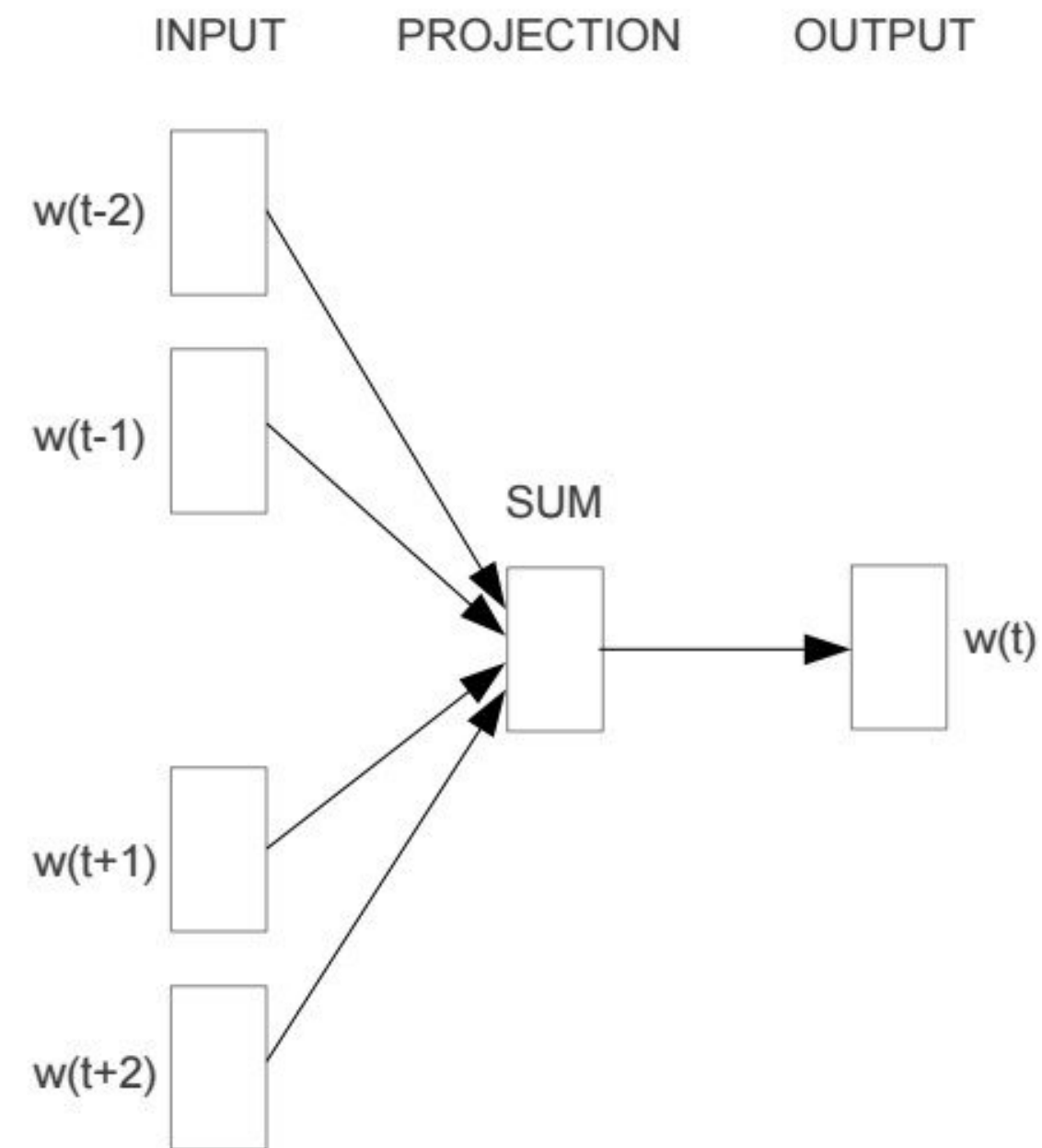
$k$  is the number of negative samples,  
and the samples follows the unigram distribution

- Other differences in factorization
  - SGNS factorizes into two unconstrained matrices (vs two orthonormal and one diagonal matrix for SVD)
  - The loss used for the factorizing is different
    - SVD - Frobenius norm (Euclidean norm / entrywise L2)
    - Word2Vec SGNS - weighted logistic loss

[Neural Word Embedding as Implicit Matrix Factorization, Levy and Goldberg, 2014]

[Improving Distributional Similarity with Lessons Learned from Word Embeddings, Levy, Goldberg, and Dagan, 2015]

# Continuous Bag of Words (CBOW)



$$L(\theta) = \prod_{t=1}^T P(w_t \mid \{w_{t+j}\}, -m \leq j \leq m, j \neq 0)$$

$$\bar{\mathbf{v}}_t = \frac{1}{2m} \sum_{-m \leq j \leq m, j \neq 0} \mathbf{v}_{t+j}$$

$$P(w_t \mid \{w_{t+j}\}) = \frac{\exp(\mathbf{u}_{w_t} \cdot \bar{\mathbf{v}}_t)}{\sum_{k \in V} \exp(\mathbf{u}_k \cdot \bar{\mathbf{v}}_t)}$$

# Skip-gram vs CBOW

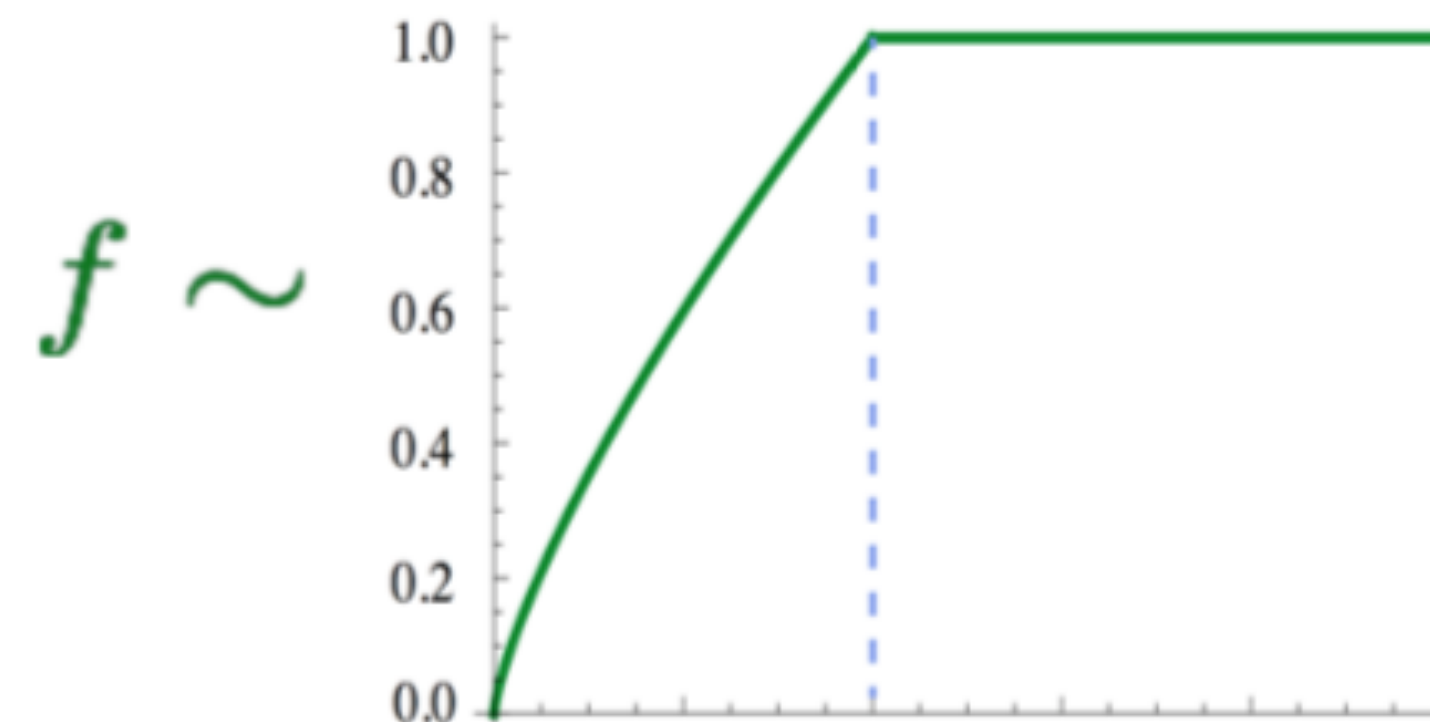
- CBOW is comparatively faster to train than skip-grams and better for frequently occurring words
- Skip-gram is slower but works well for smaller amount of data and works well for less frequently occurring words
- CBOW is a simpler problem than Skip-gram because in CBOW we just need to predict the one center word given many context words

# GloVe: **G**lobal **V**ectors

- Let's take the global co-occurrence statistics:  $X_{i,j}$
- Try to learn word vectors to predict the co-occurrence counts (using L2 loss)
- Function  $f$  to weight loss by frequency of words (from 0 to 1)

$$J = \sum_{i,j=1}^{|V|} f(X_{ij})(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij})^2$$

- Final word vector:  $w_i + \tilde{w}_j$
- Training faster
- Scalable to very large corpora



(Pennington et al, 2014): GloVe: Global Vectors for Word Representation

# GloVe: **G**lobal **V**ectors

Nearest words to  
**frog:**

1. frogs
2. toad
3. litoria
4. leptodactylidae
5. rana
6. lizard
7. eleutherodactylus



litoria



leptodactylidae



rana



eleutherodactylus

(Pennington et al, 2014): GloVe: Global Vectors for Word Representation

# FastText: Sub-Word Embeddings

- Similar as Skip-gram, but break words into n-grams with  $n = 3$  to  $6$

where: 3-grams: <wh, whe, her, ere, re>

4-grams: <whe, wher, here, ere>

5-grams: <wher, where, here>

6-grams: <where, where>

Note: All the embeddings that we have learned are also called “static word embeddings”: there is one fixed vector for every word in the vocabulary.

- Replace  $\mathbf{u}_i \cdot \mathbf{v}_j$  by  $\sum_{g \in n\text{-grams}(w_i)} \mathbf{u}_g \cdot \mathbf{v}_j$

- More to come! Contextualized word embeddings



(Bojanowski et al, 2017): Enriching Word Vectors with Subword Information

# Trained word embeddings available

- word2vec: <https://code.google.com/archive/p/word2vec/>
- GloVe: <https://nlp.stanford.edu/projects/glove/>
- FastText: <https://fasttext.cc/>

## Download pre-trained word vectors

- Pre-trained word vectors. This data is made available under the [Public Domain Dedication and License](http://www.opendatacommons.org/licenses/pddl/1.0/) v1.0 whose full text can be found at: <http://www.opendatacommons.org/licenses/pddl/1.0/>.
  - [Wikipedia 2014](#) + [Gigaword 5](#) (6B tokens, 400K vocab, uncased, 50d, 100d, 200d, & 300d vectors, 822 MB download): [glove.6B.zip](#)
  - Common Crawl (42B tokens, 1.9M vocab, uncased, 300d vectors, 1.75 GB download): [glove.42B.300d.zip](#)
  - Common Crawl (840B tokens, 2.2M vocab, cased, 300d vectors, 2.03 GB download): [glove.840B.300d.zip](#)
  - Twitter (2B tweets, 27B tokens, 1.2M vocab, uncased, 25d, 50d, 100d, & 200d vectors, 1.42 GB download): [glove.twitter.27B.zip](#)
- Ruby [script](#) for preprocessing Twitter data

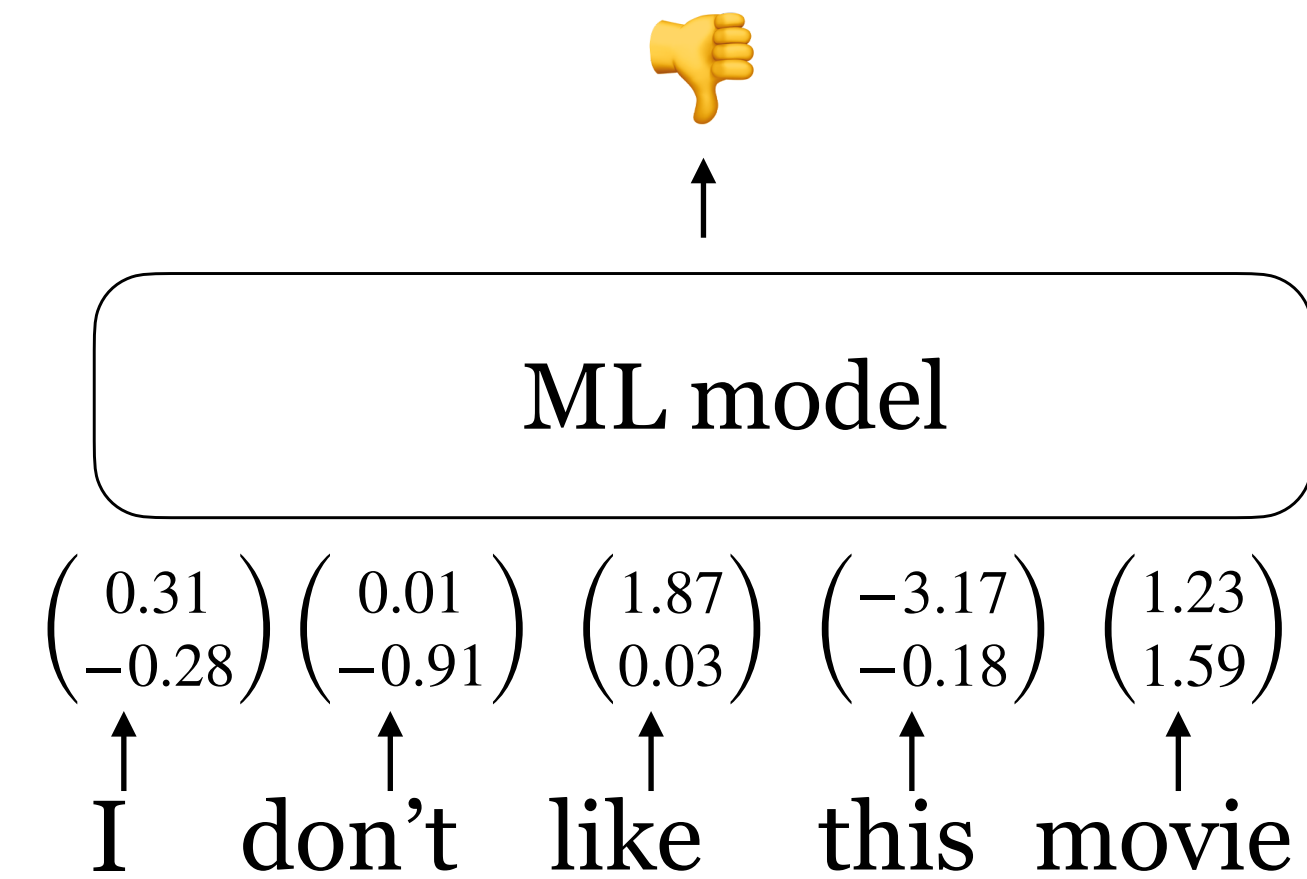
Differ in algorithms, text corpora, dimensions, cased/uncased...

# Evaluating Word Embeddings

# Extrinsic vs intrinsic evaluation

## Extrinsic evaluation

- Let's plug these word embeddings into a real NLP system and see whether this improves performance
- Could take a long time but still the most important evaluation metric



## Intrinsic evaluation

- Evaluate on a specific/intermediate subtask
- Fast to compute
- Not clear if it really helps the downstream task

# Intrinsic evaluation

## Word similarity

Example dataset: wordsim-353

353 pairs of words with human judgement

<http://www.cs.technion.ac.il/~gabr/resources/data/wordsim353/>

| Word 1    | Word 2   | Human (mean) |
|-----------|----------|--------------|
| tiger     | cat      | 7.35         |
| tiger     | tiger    | 10           |
| book      | paper    | 7.46         |
| computer  | internet | 7.58         |
| plane     | car      | 5.77         |
| professor | doctor   | 6.62         |
| stock     | phone    | 1.62         |
| stock     | CD       | 1.31         |
| stock     | jaguar   | 0.92         |

Cosine similarity:

$$\cos(\mathbf{u}_i, \mathbf{u}_j) = \frac{\mathbf{u}_i \cdot \mathbf{u}_j}{\|\mathbf{u}_i\|_2 \times \|\mathbf{u}_j\|_2}$$

Metric: Spearman rank correlation  
(Pearson correlation of ranks)

$$r_s = \rho_{R(X), R(Y)} = \frac{\text{cov}(R(X), R(Y))}{\sigma_{R(X)} \sigma_{R(Y)}}$$

# Intrinsic evaluation

## Word Similarity

| Model             | Size | WS353              | MC                 | RG                 | SCWS               | RW                 |
|-------------------|------|--------------------|--------------------|--------------------|--------------------|--------------------|
| SVD               | 6B   | 35.3               | 35.1               | 42.5               | 38.3               | 25.6               |
| SVD-S             | 6B   | 56.5               | 71.5               | 71.0               | 53.6               | 34.7               |
| SVD-L             | 6B   | 65.7               | <u>72.7</u>        | 75.1               | 56.5               | 37.0               |
| CBOW <sup>†</sup> | 6B   | 57.2               | 65.6               | 68.2               | 57.0               | 32.5               |
| SG <sup>†</sup>   | 6B   | 62.8               | 65.2               | 69.7               | <u>58.1</u>        | 37.2               |
| GloVe             | 6B   | <u>65.8</u>        | <u>72.7</u>        | <u>77.8</u>        | 53.9               | <u>38.1</u>        |
| SVD-L             | 42B  | 74.0               | 76.4               | 74.1               | 58.3               | 39.9               |
| GloVe             | 42B  | <b><u>75.9</u></b> | <b><u>83.6</u></b> | <b><u>82.9</u></b> | <b><u>59.6</u></b> | <b><u>47.8</u></b> |
| CBOW*             | 100B | 68.4               | 79.6               | 75.4               | 59.4               | 45.5               |

- Spearman rank correlation of word vector similarities with different human judgements on different datasets
- All vectors are 300-dimensional

(Pennington et al, 2014): GloVe: Global Vectors for Word Representation

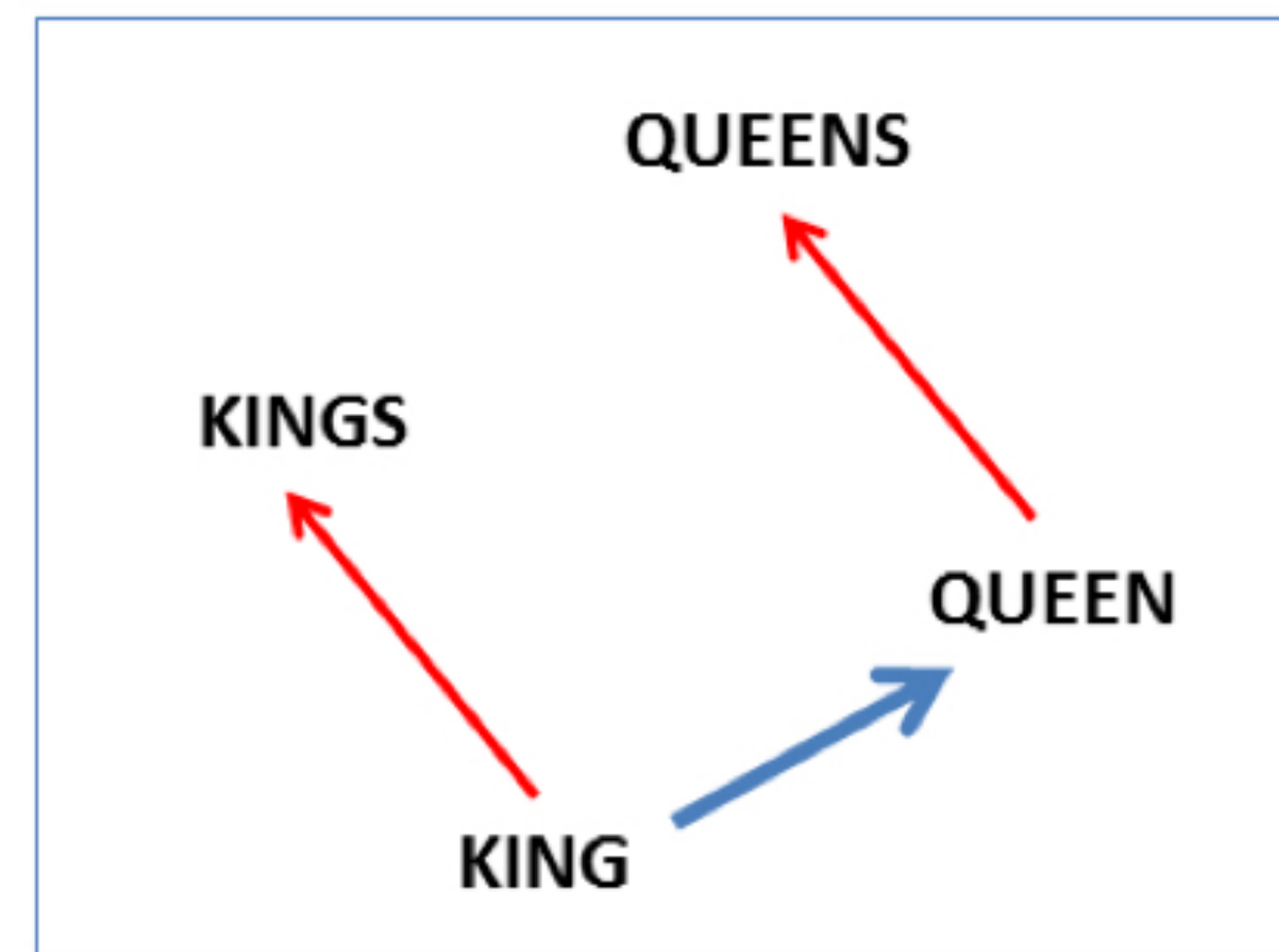
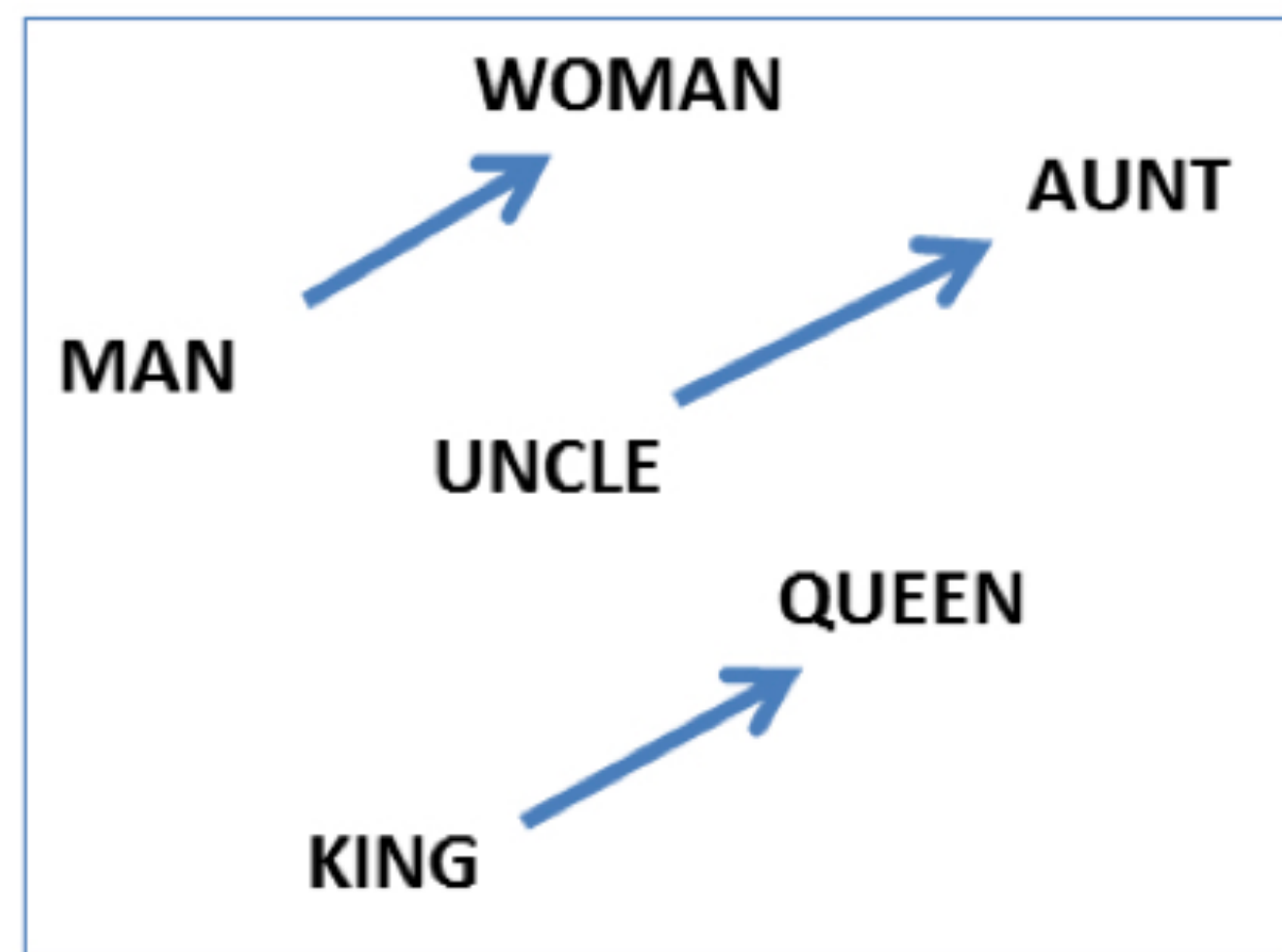
# Intrinsic evaluation

## Word analogy

Analogy: Embeddings capture relational meaning!

$\text{vector}('king') - \text{vector}('man') + \text{vector}('woman') \approx \text{vector}('queen')$

$\text{vector}('Paris') - \text{vector}('France') + \text{vector}('Italy') \approx \text{vector}('Rome')$



# Intrinsic evaluation

## Word analogy

man: woman  $\approx$  king: ?

$$\arg \max_i (\cos(\mathbf{u}_i, \mathbf{u}_b - \mathbf{u}_a + \mathbf{u}_c))$$

semantic

Chicago: Illinois  $\approx$  Philadelphia: ?

syntactic

bad: worst  $\approx$  cool: ?

More examples at

<http://download.tensorflow.org/data/questions-words.txt>

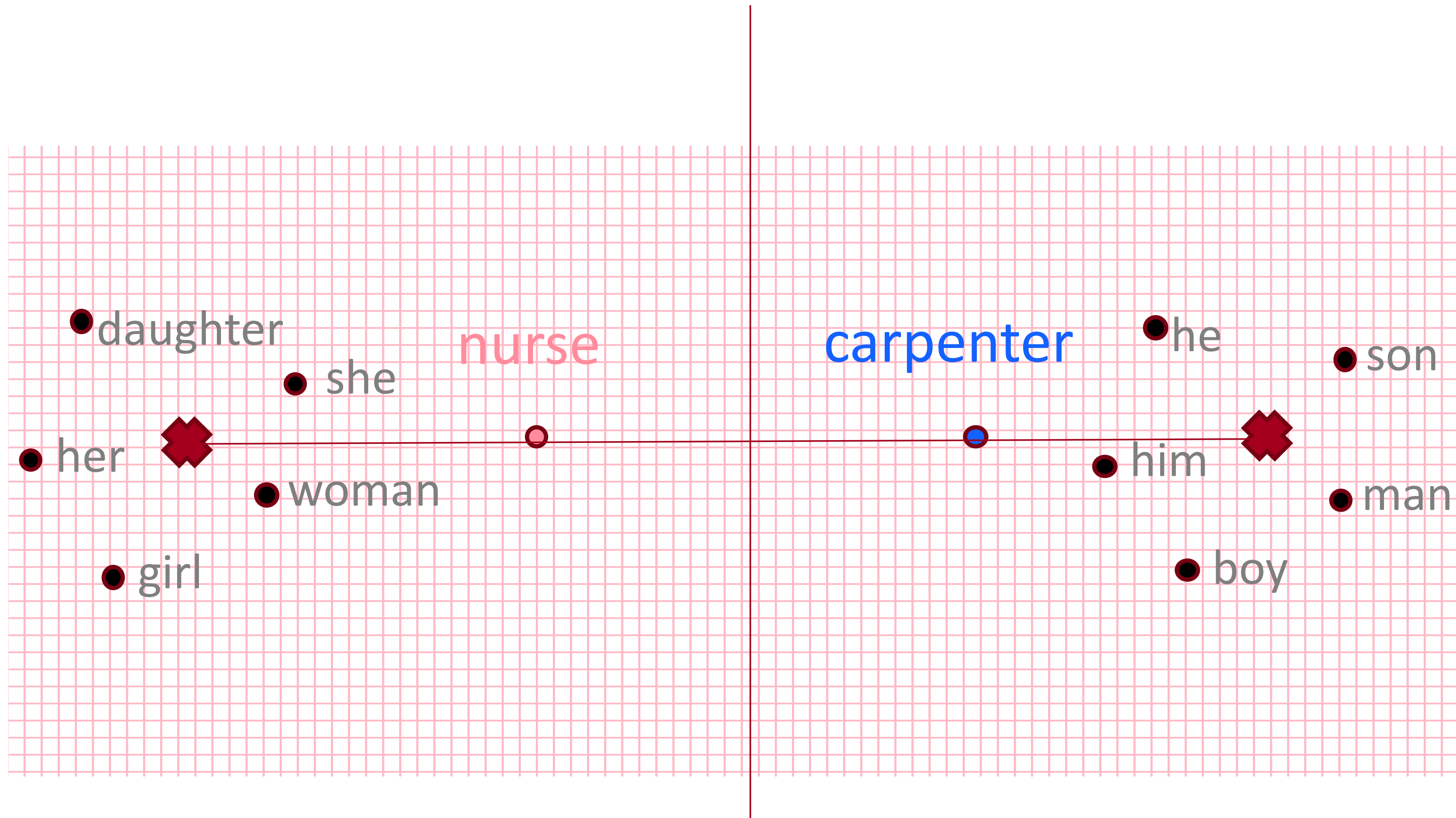
| Model             | Dim. | Size | Sem.               | Syn.               | Tot.               |
|-------------------|------|------|--------------------|--------------------|--------------------|
| ivLBL             | 100  | 1.5B | 55.9               | 50.1               | 53.2               |
| HPCA              | 100  | 1.6B | 4.2                | 16.4               | 10.8               |
| GloVe             | 100  | 1.6B | <u>67.5</u>        | <u>54.3</u>        | <u>60.3</u>        |
| SG                | 300  | 1B   | 61                 | 61                 | 61                 |
| CBOW              | 300  | 1.6B | 16.1               | 52.6               | 36.1               |
| vLBL              | 300  | 1.5B | 54.2               | <u>64.8</u>        | 60.0               |
| ivLBL             | 300  | 1.5B | 65.2               | 63.0               | 64.0               |
| GloVe             | 300  | 1.6B | <u>80.8</u>        | 61.5               | <u>70.3</u>        |
| SVD               | 300  | 6B   | 6.3                | 8.1                | 7.3                |
| SVD-S             | 300  | 6B   | 36.7               | 46.6               | 42.1               |
| SVD-L             | 300  | 6B   | 56.6               | 63.0               | 60.1               |
| CBOW <sup>†</sup> | 300  | 6B   | 63.6               | <u>67.4</u>        | 65.7               |
| SG <sup>†</sup>   | 300  | 6B   | 73.0               | 66.0               | 69.1               |
| GloVe             | 300  | 6B   | <u>77.4</u>        | 67.0               | <u>71.7</u>        |
| CBOW              | 1000 | 6B   | 57.3               | 68.9               | 63.7               |
| SG                | 1000 | 6B   | 66.1               | 65.1               | 65.6               |
| SVD-L             | 300  | 42B  | 38.4               | 58.2               | 49.2               |
| GloVe             | 300  | 42B  | <b><u>81.9</u></b> | <b><u>69.3</u></b> | <b><u>75.0</u></b> |

# Hyperparameters and settings matter

| Word Similarity |          |                       |                        |                     |                            |                            |                       | Analogies           |                    |
|-----------------|----------|-----------------------|------------------------|---------------------|----------------------------|----------------------------|-----------------------|---------------------|--------------------|
| win             | Method   | WordSim<br>Similarity | WordSim<br>Relatedness | Bruni et al.<br>MEN | Radinsky et al.<br>M. Turk | Luong et al.<br>Rare Words | Hill et al.<br>SimLex | Google<br>Add / Mul | MSR<br>Add / Mul   |
| 2               | PPMI     | .732                  | <b>.699</b>            | .744                | .654                       | .457                       | .382                  | .552 / .677         | .306 / .535        |
|                 | SVD      | .772                  | .671                   | <b>.777</b>         | .647                       | <b>.508</b>                | .425                  | .554 / .591         | .408 / .468        |
|                 | SGNS     | <b>.789</b>           | .675                   | .773                | <b>.661</b>                | .449                       | <b>.433</b>           | .676 / <b>.689</b>  | .617 / <b>.644</b> |
|                 | GloVe    | .720                  | .605                   | .728                | .606                       | .389                       | .388                  | .649 / .666         | .540 / .591        |
| 5               | PPMI     | .732                  | <b>.706</b>            | .738                | <b>.668</b>                | .442                       | .360                  | .518 / .649         | .277 / .467        |
|                 | SVD      | .764                  | .679                   | <b>.776</b>         | .639                       | <b>.499</b>                | <b>.416</b>           | .532 / .569         | .369 / .424        |
|                 | SGNS     | <b>.772</b>           | .690                   | .772                | .663                       | .454                       | .403                  | .692 / <b>.714</b>  | .605 / <b>.645</b> |
|                 | GloVe    | .745                  | .617                   | .746                | .631                       | .416                       | .389                  | .700 / .712         | .541 / .599        |
| 10              | PPMI     | .735                  | <b>.701</b>            | .741                | .663                       | .235                       | .336                  | .532 / .605         | .249 / .353        |
|                 | SVD      | .766                  | .681                   | .770                | .628                       | <b>.312</b>                | .419                  | .526 / .562         | .356 / .406        |
|                 | SGNS     | <b>.794</b>           | .700                   | <b>.775</b>         | <b>.678</b>                | .281                       | <b>.422</b>           | .694 / .710         | .520 / <b>.557</b> |
|                 | GloVe    | .746                  | .643                   | .754                | .616                       | .266                       | .375                  | .702 / <b>.712</b>  | .463 / .519        |
| 10              | SGNS-LS  | .766                  | .681                   | <b>.781</b>         | <b>.689</b>                | <b>.451</b>                | .414                  | .739 / <b>.758</b>  | .690 / <b>.729</b> |
|                 | GloVe-LS | .678                  | .624                   | .752                | .639                       | .361                       | .371                  | .732 / .750         | .628 / .685        |

[Improving Distributional Similarity with Lessons Learned from Word Embeddings, Levy, Goldberg, and Dagan, 2015]

# Bias in word embeddings

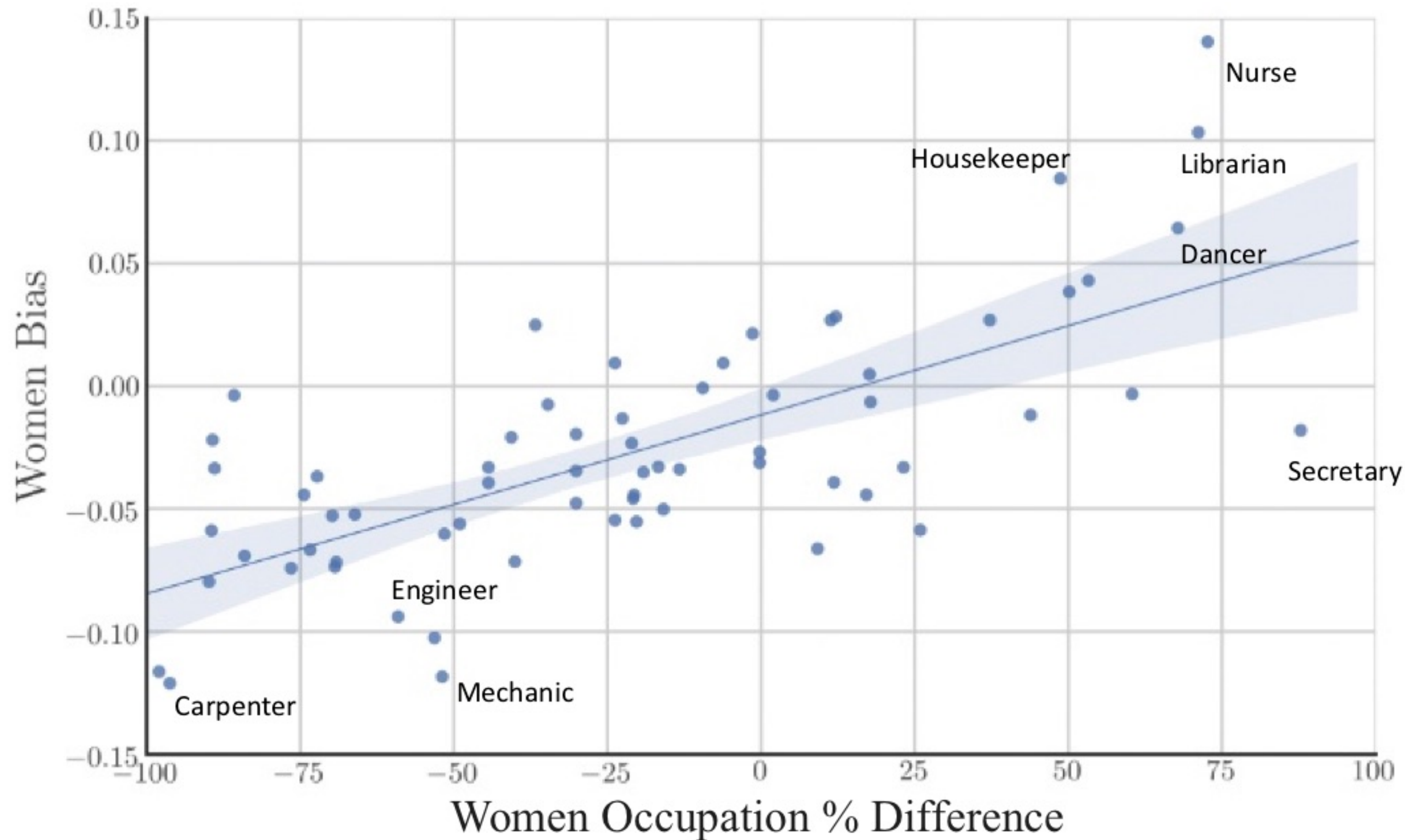


Embeddings reflect cultural bias

Biases change over time

# Embedding bias correlates with actual occupation data

Word embeddings quantify 100 years of gender and ethnic stereotypes, Garg et al, 2018



# Beyond Simple Word Embeddings

# Extensions to word vectors

- Subword embeddings (FastText)
- Phrases and multi-word expressions
- Sense embeddings
- Embeddings using other types of context, spaces
- Cross-lingual and cross-modal embeddings
- Context dependent embeddings (Elmo, BERT)

# Simple sentence embeddings from word embeddings

- Take average

$$v_s = \frac{1}{|s|} \sum_{w \in s} v_w$$

- Take weighted average

$$v_s = \frac{1}{|s|} \sum_{w \in s} \frac{a}{a + p(w)} v_w$$

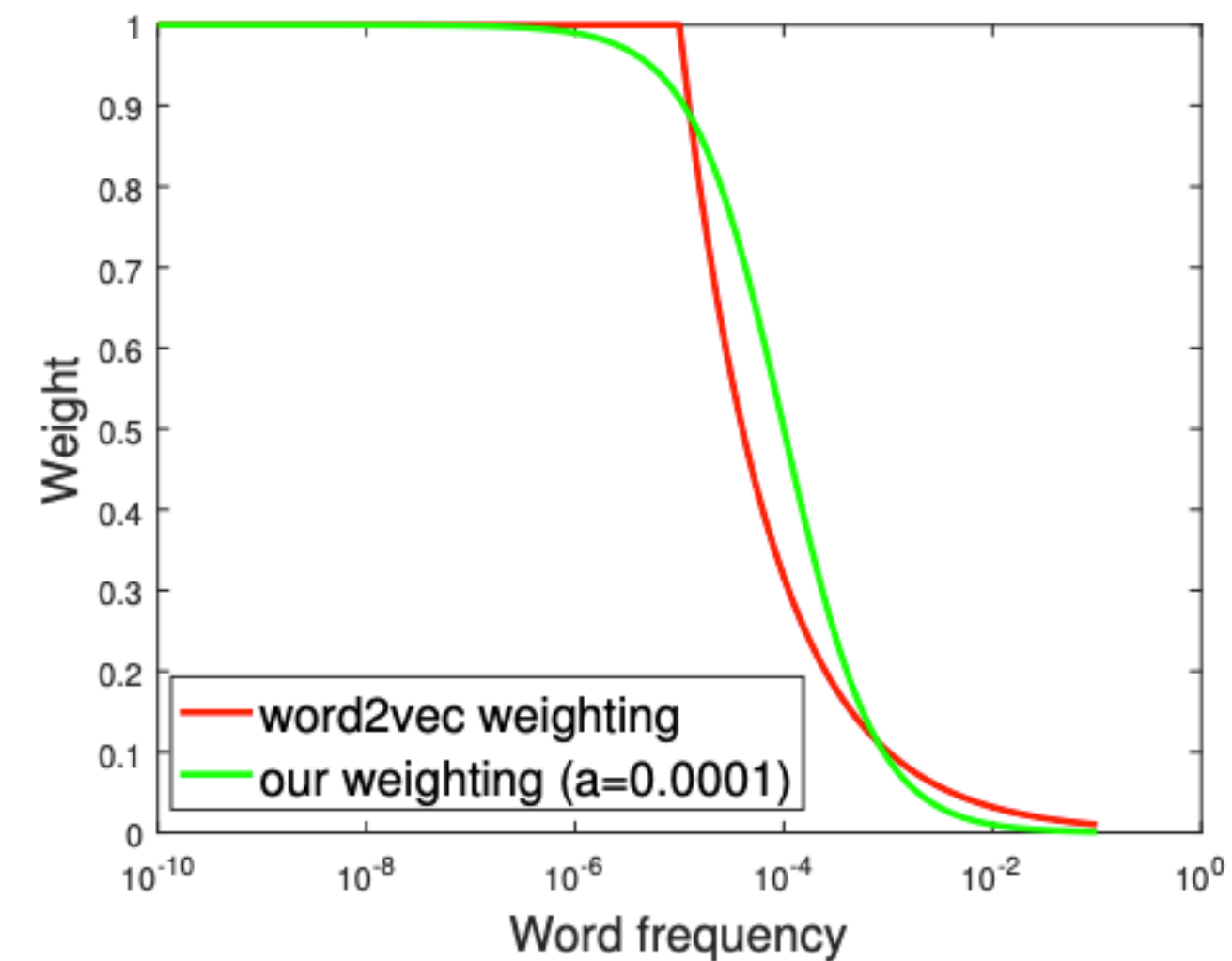
Weight rare words more

- Shift sentence embedding

Let  $V_s$  be matrix whose columns are the sentence vectors  $v_s$ ,  
and  $u$  be the first singular vector of  $V_s$

$$v'_s = v_s - uu^\top v_s$$

Basic idea: try to subtract out vector corresponding to syntax  
(assume it is vector associated with first singular value)

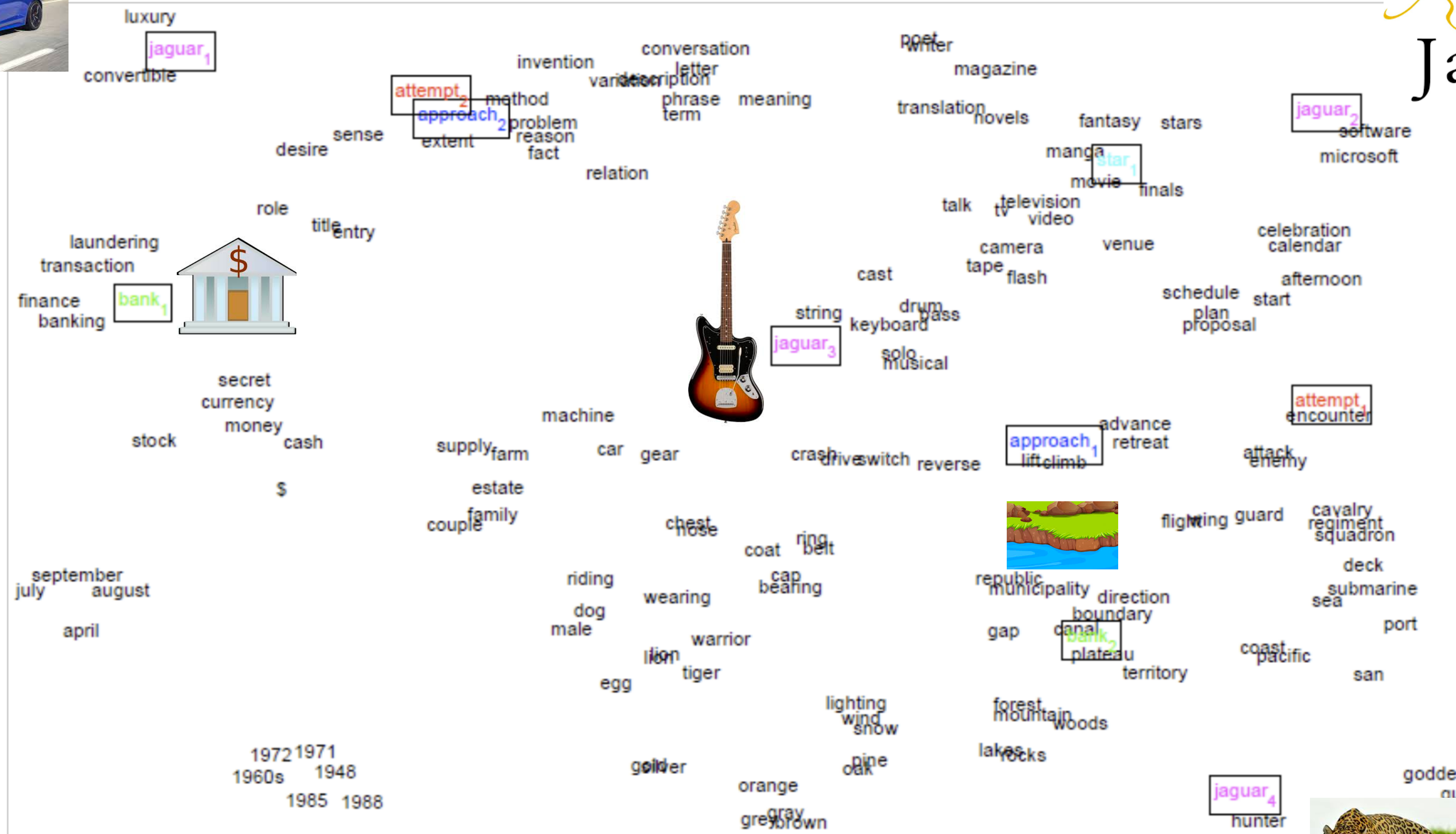


All these are BOW models

Later we will look at models (RNNs)  
that take order into account.

[A simple but tough-to-beat baseline for sentence embeddings, Arora, Liang and Ma, 2017]

# Sense Embeddings

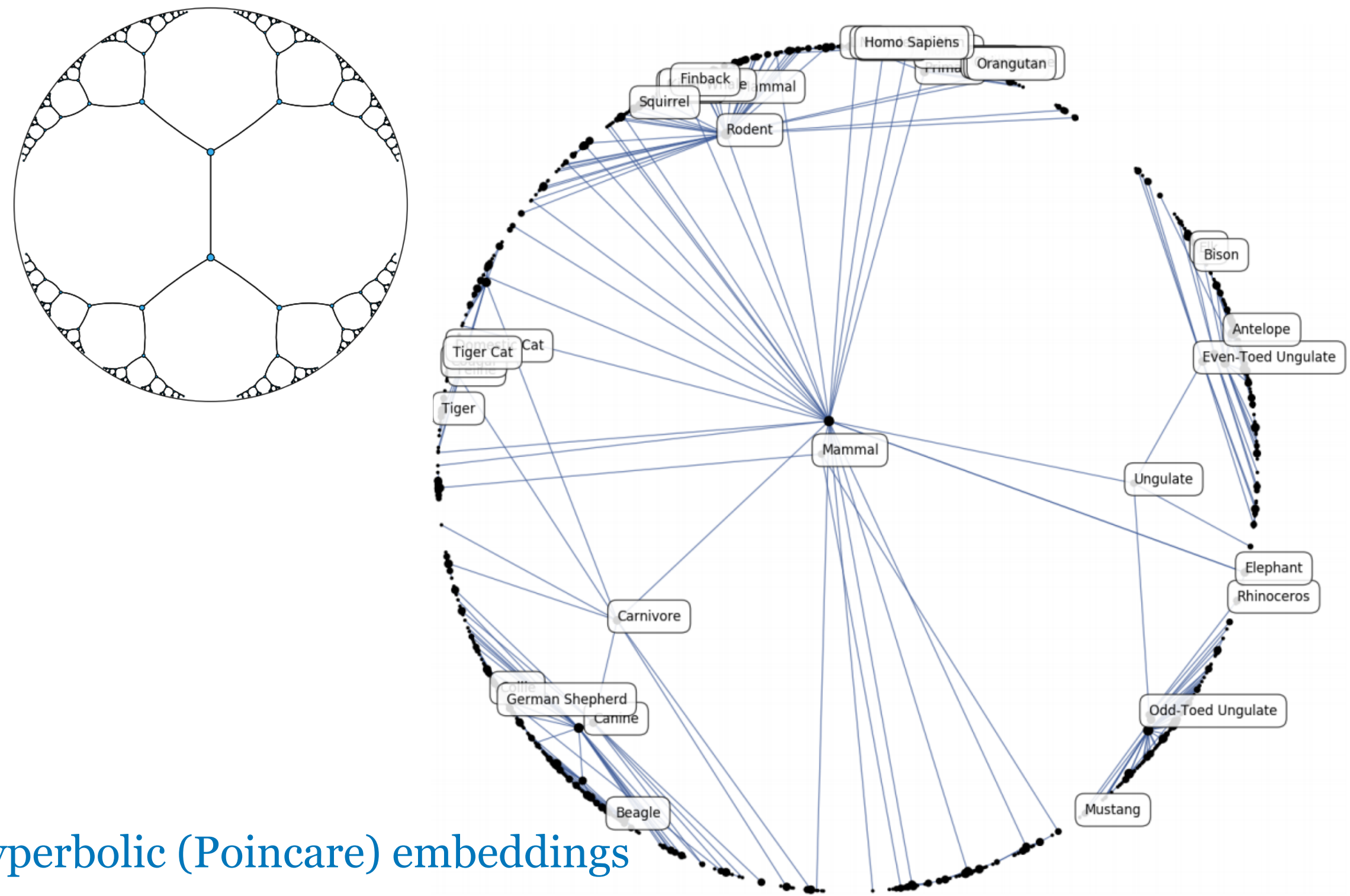


Improving Word Representations via Global Context and Multiple Word Prototypes [Huang et al, 2012]



# Beyond Euclidean spaces

Tree with equally spaced nodes

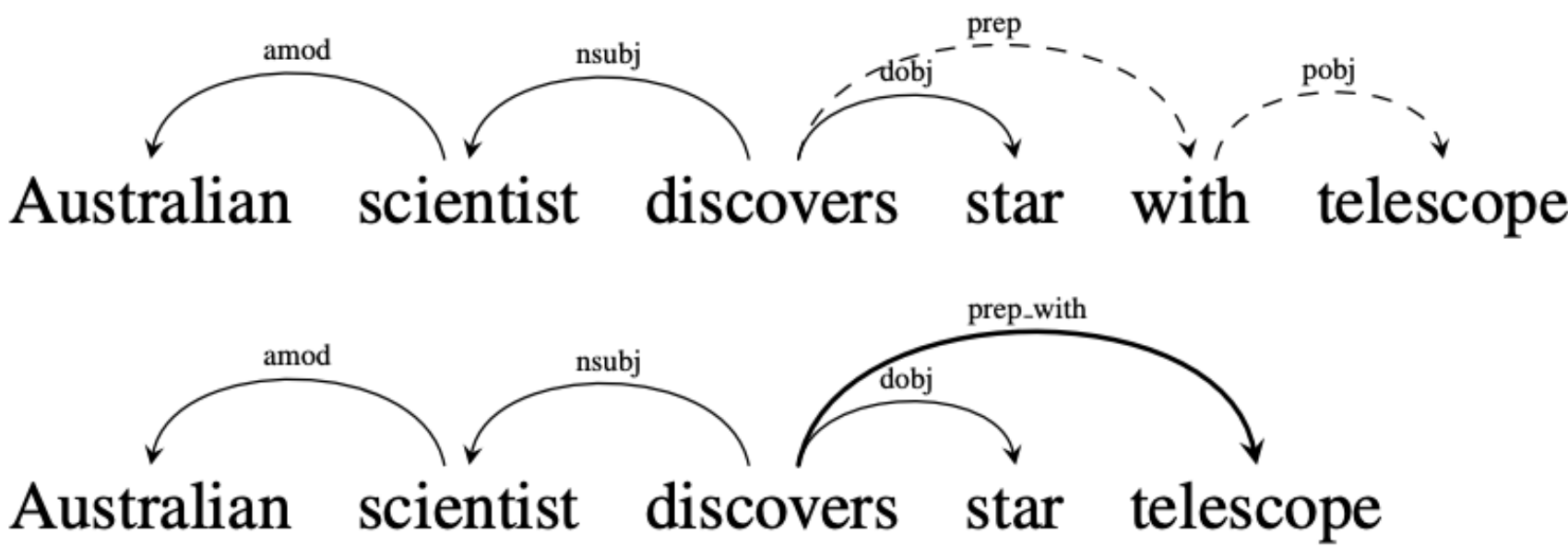


## Hyperbolic (Poincare) embeddings

- Distances grow exponentially as points go toward the boundary
- better for hierarchically organized

Poincaré Embeddings for Learning Hierarchical Representations  
[Nickel and Kiela, 2017]

# Syntax as context

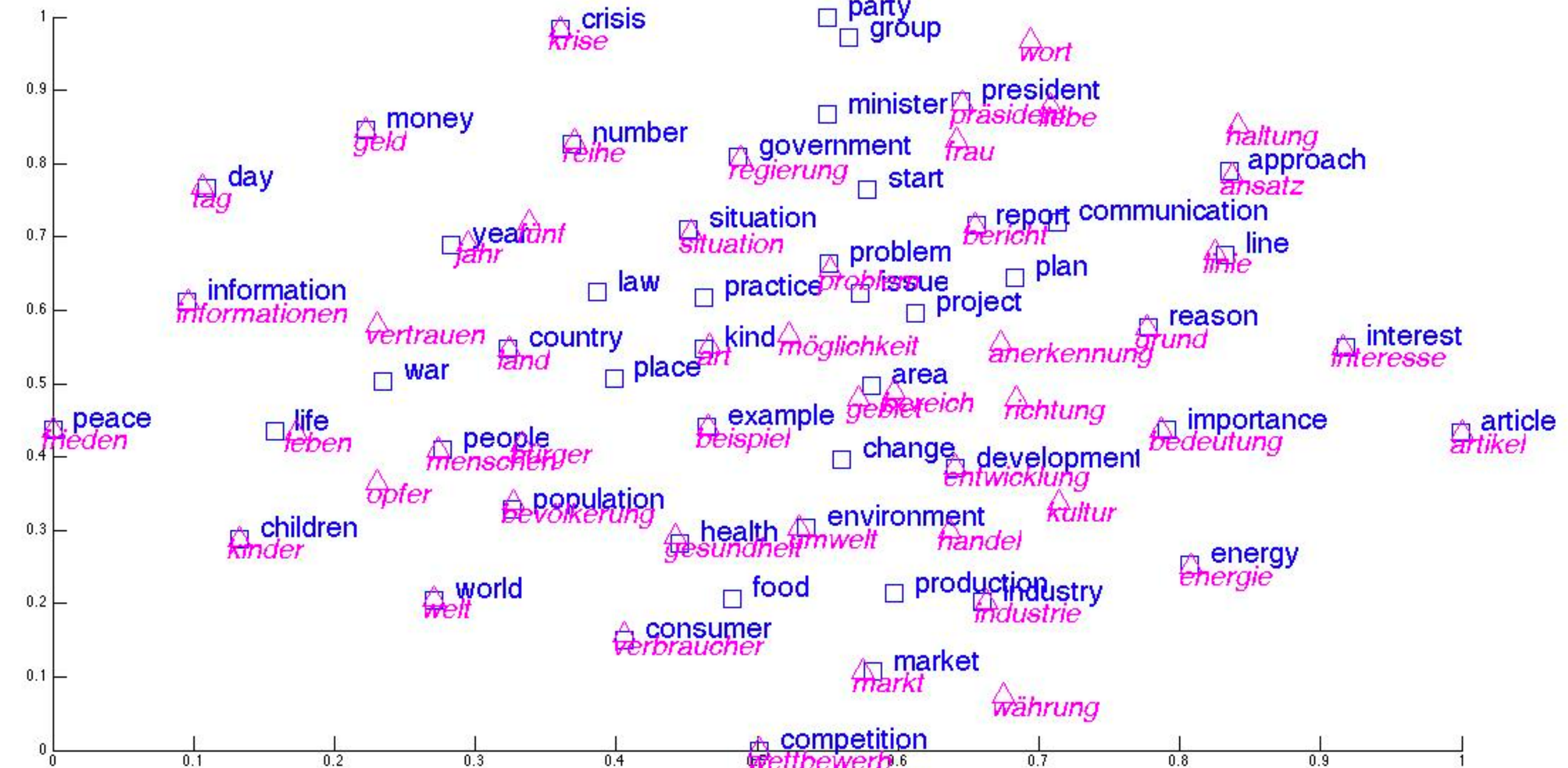
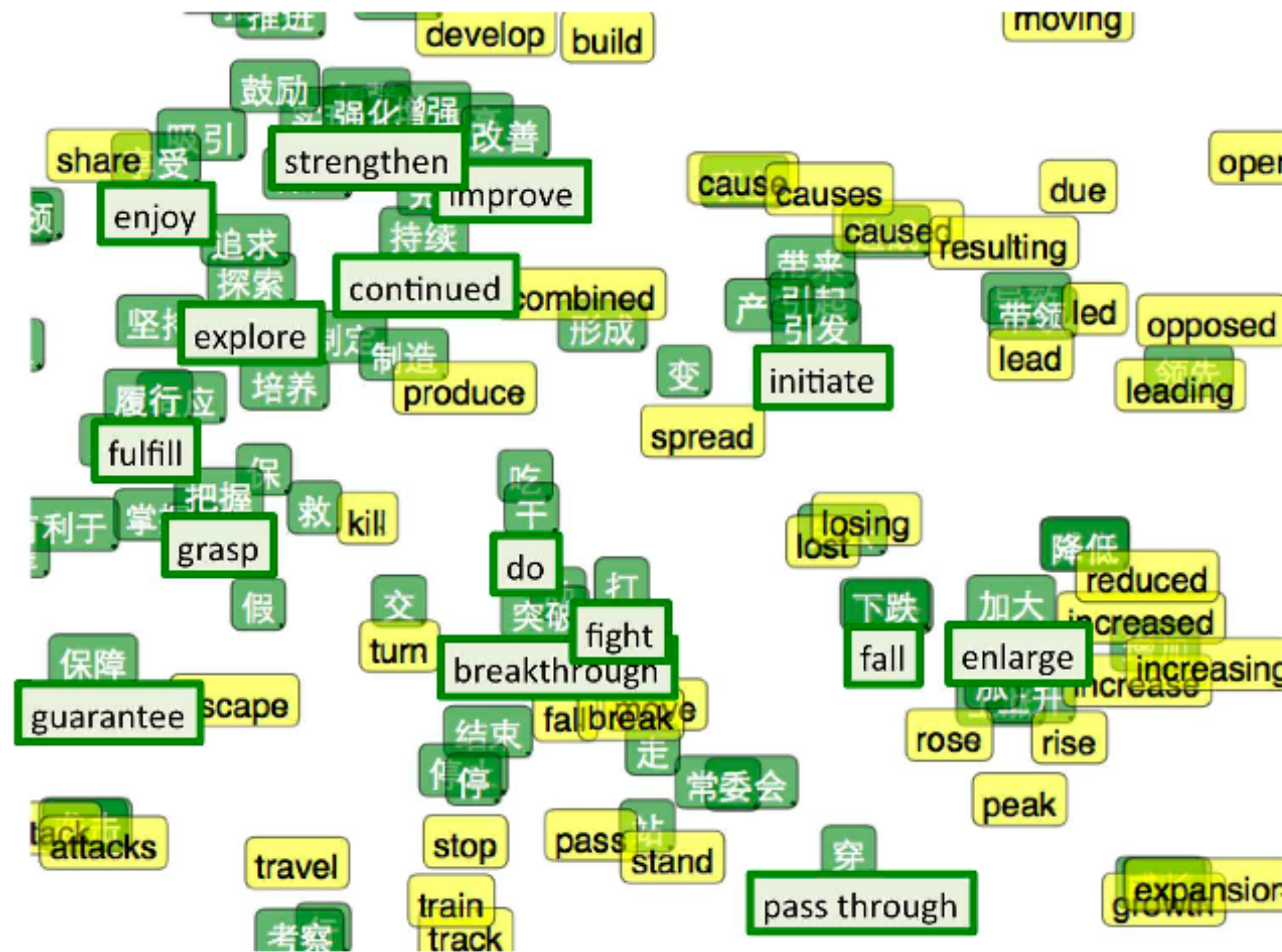


| WORD       | CONTEXTS                                        |
|------------|-------------------------------------------------|
| australian | scientist/amod <sup>-1</sup>                    |
| scientist  | australian/amod, discovers/nsubj <sup>-1</sup>  |
| discovers  | scientist/nsubj, star/dobj, telescope/prep_with |
| star       | discovers/dobj <sup>-1</sup>                    |
| telescope  | discovers/prep_with <sup>-1</sup>               |

## Use dependency parse neighbors as context

Dependency based word embeddings  
[Levy and Goldberg, 2014]

# Cross-lingual word embeddings



See <https://ruder.io/cross-lingual-embeddings/>

Bilingual Word Representations with Monolingual Quality in Mind  
[Luong et al, 2015]

# Cross-modal embeddings

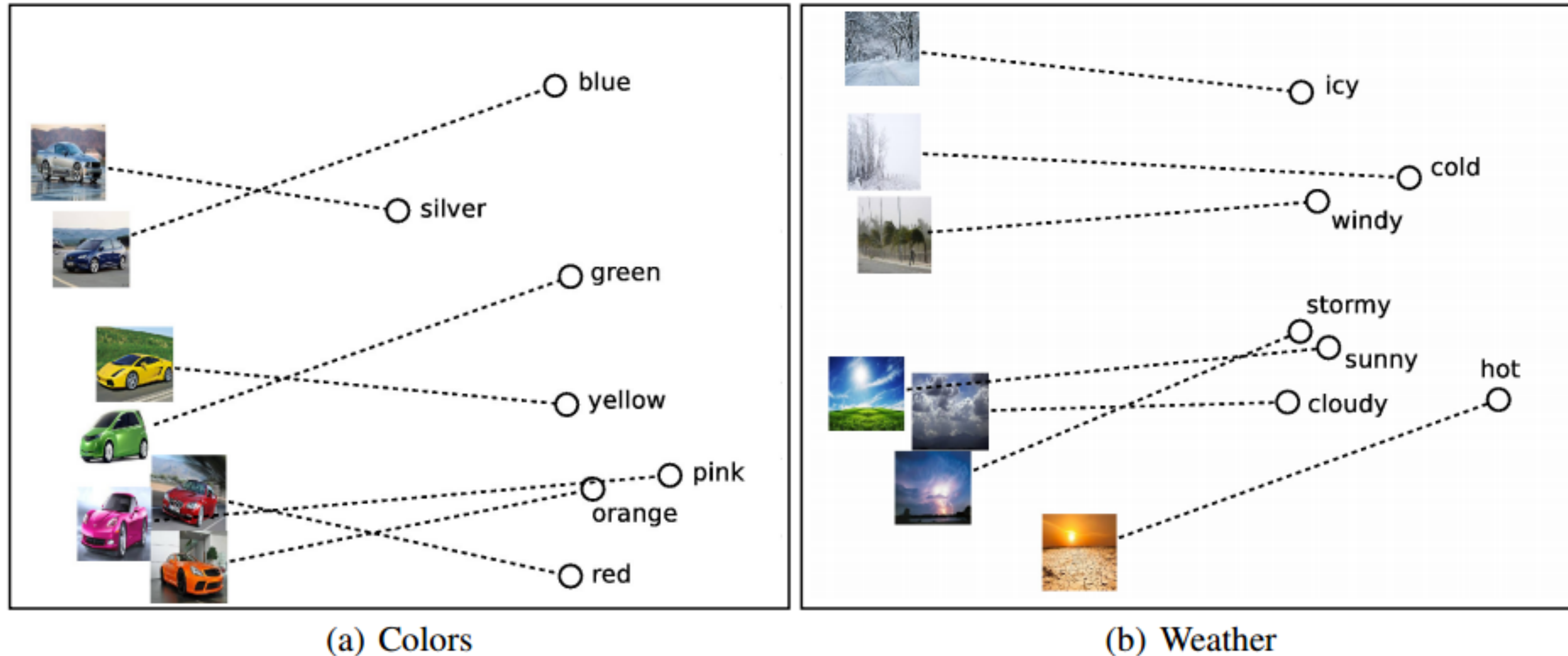


Figure 5: PCA projection of the 300-dimensional word and image representations for (a) cars and colors and (b) weather and temperature.

# Summary

- Word embeddings can be learned using Word2Vec and GloVe
- You can also use SVD on PPMI co-occurrence matrix to obtain dense word vectors
- There is connection between Word2Vec and PPMI co-occurrence matrix
- Evaluating embeddings
- Bias in embeddings
- Some ways to extend embeddings