



CMPT 825: Natural Language Processing

Word Embeddings - Word2Vec

Fall 2020
2020-09-30

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

Announcements

- Homework 1 due today
 - Both parts are due
 - Programming component has 2 grace days, but something must be turned in by tonight
 - Single person groups - highly encouraged to team up with each other
- Video Lectures
 - Summary of logistic regression (optional)
 - Word vectors (required) - covers PPMI
 - Word vectors TF-IDF (required, not yet posted) - covers TF-IDF
 - Word vectors Summary (optional, not yet posted)
 - Using SVD to get dense word vectors, and connections to word2vec
- TA video summarizing key points about word vectors

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**.

Word Vectors

- One-hot vectors

hotel = [0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0]

motel = [0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0]

- Represent words by their context

word-word (term-context) co-occurrence matrix

term →

	aardvark	computer	data	pinch	result	sugar	...
apricot	0	0	0	1	0	1	
pineapple	0	0	0	1	0	1	
digital	0	2	1	0	1	0	
information	0	1	6	0	4	0	

context
(other words in the span
around the target word)

$|V| \times |V|$ matrix

sugar, a sliced lemon, a tablespoonful of
their enjoyment. Cautiously she sampled her first
well suited to programming on the digital
for the purpose of gathering data and

apricot
pineapple
computer.
information

jam, a pinch each of,
and another fruit whose taste she likened
In finding the optimal R-stage policy from
necessary for the study authorized in the

Sparse vs dense vectors

Vectors we get from word-word (term-context) co-occurrence matrix are

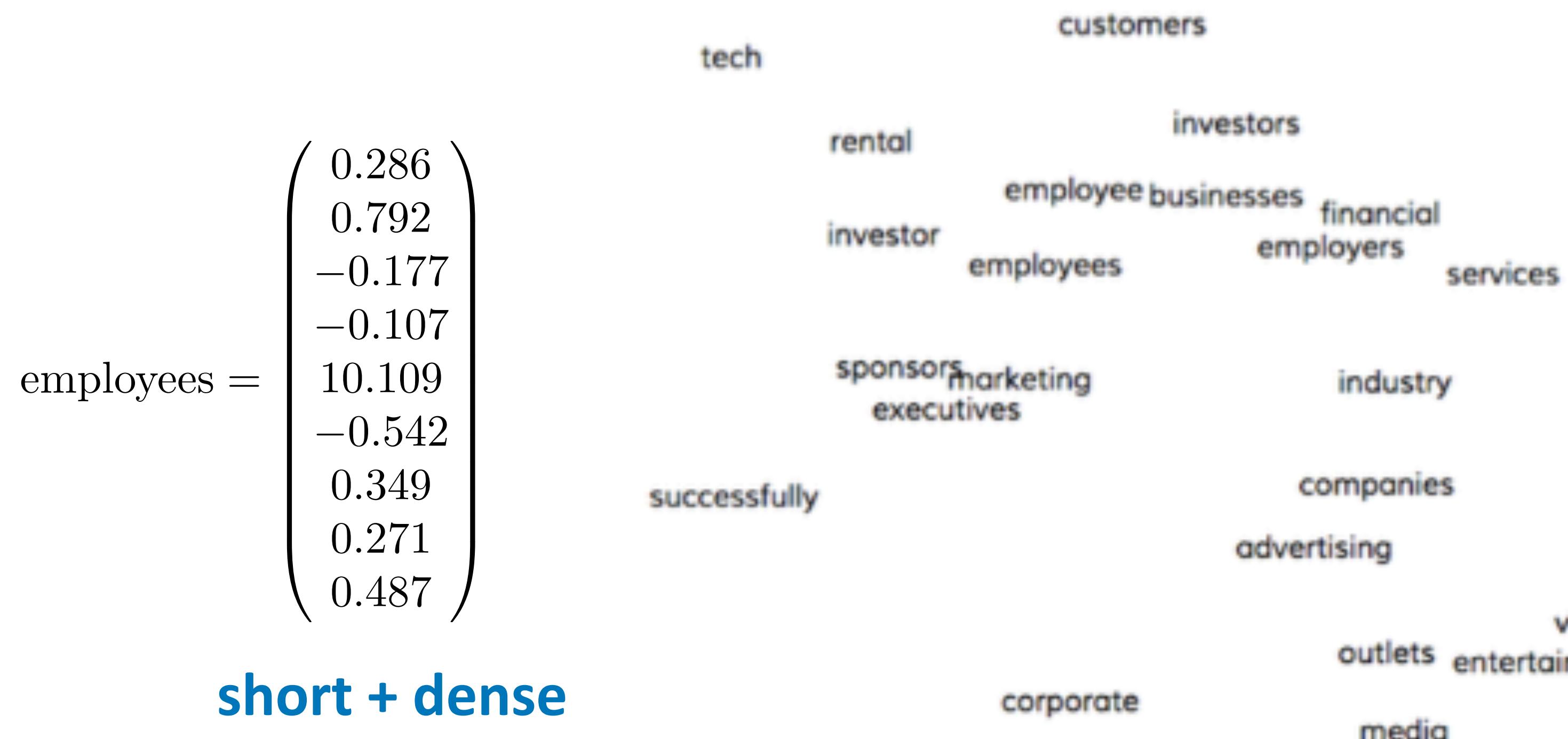
- **long** (length $|V| = 20,000$ to $50,000$)
- **sparse** (most elements are zero)

True for both one-hot, tf-idf and PPMI vectors

Alternative: we want to represent words as

- **short** (50-300 dimensional)
- **dense** (real-valued) vectors
- The focus of this lecture
- The basis of all the modern NLP systems

Dense vectors



Why 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!

$$\text{SVD} \quad \begin{matrix} \text{word-word} \\ \text{PPMI matrix} \\ X \end{matrix}_{w \times c} = \begin{matrix} W \end{matrix}_{w \times m} \begin{matrix} \Sigma \end{matrix}_{m \times m} \begin{matrix} C \end{matrix}_{m \times c}$$

Word2vec and friends

Download pretrained word embeddings

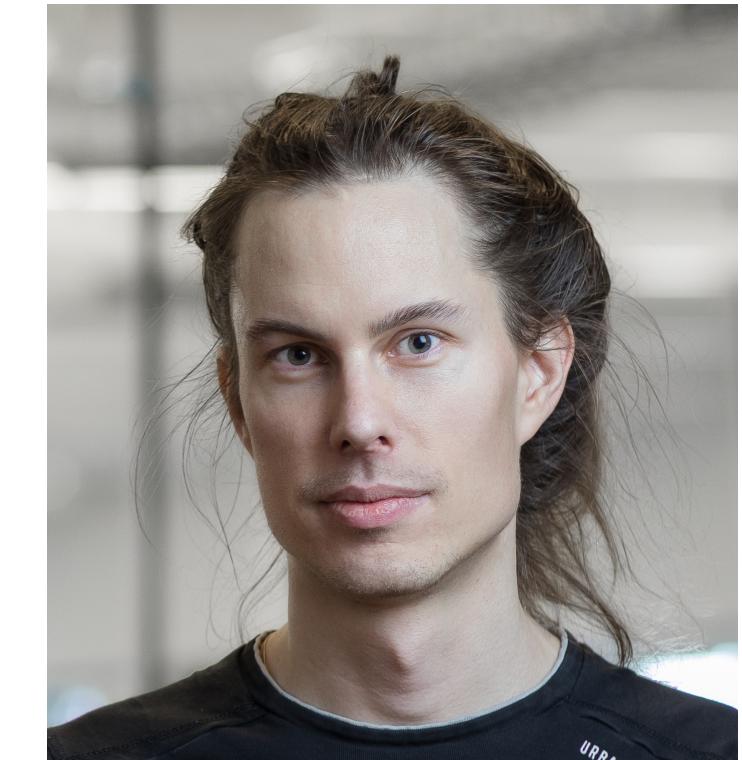
Google

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**

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



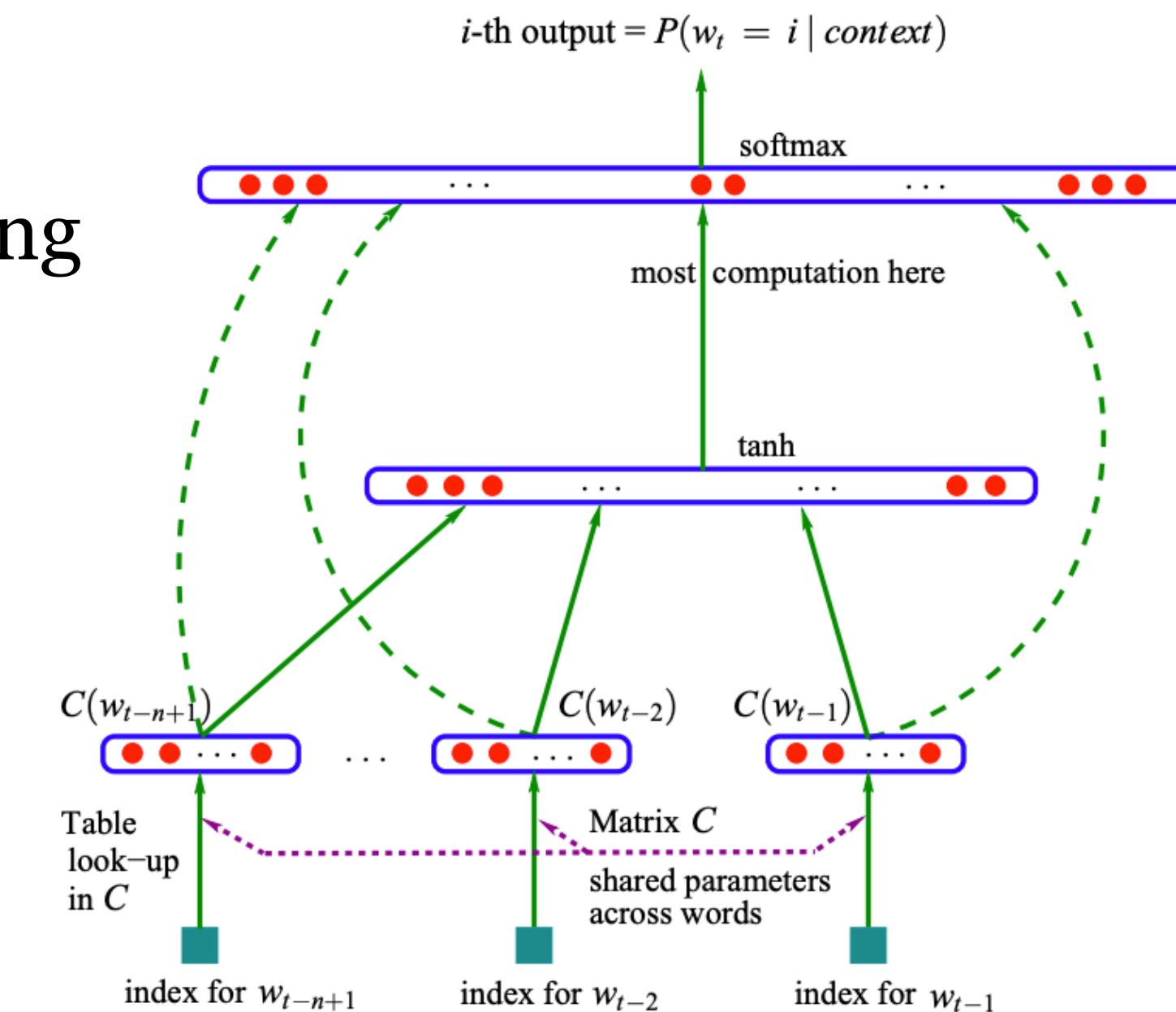
Word2Vec

- 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**

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:
 - Wikipedia + Gigaword 5: 6B
 - Twitter: 27B
 - Common Crawl: 840B

- Output: $f : V \rightarrow \mathbb{R}^d$

$$v_{\text{cat}} = \begin{pmatrix} -0.224 \\ 0.130 \\ -0.290 \\ 0.276 \end{pmatrix}$$

$$v_{\text{dog}} = \begin{pmatrix} -0.124 \\ 0.430 \\ -0.200 \\ 0.329 \end{pmatrix}$$

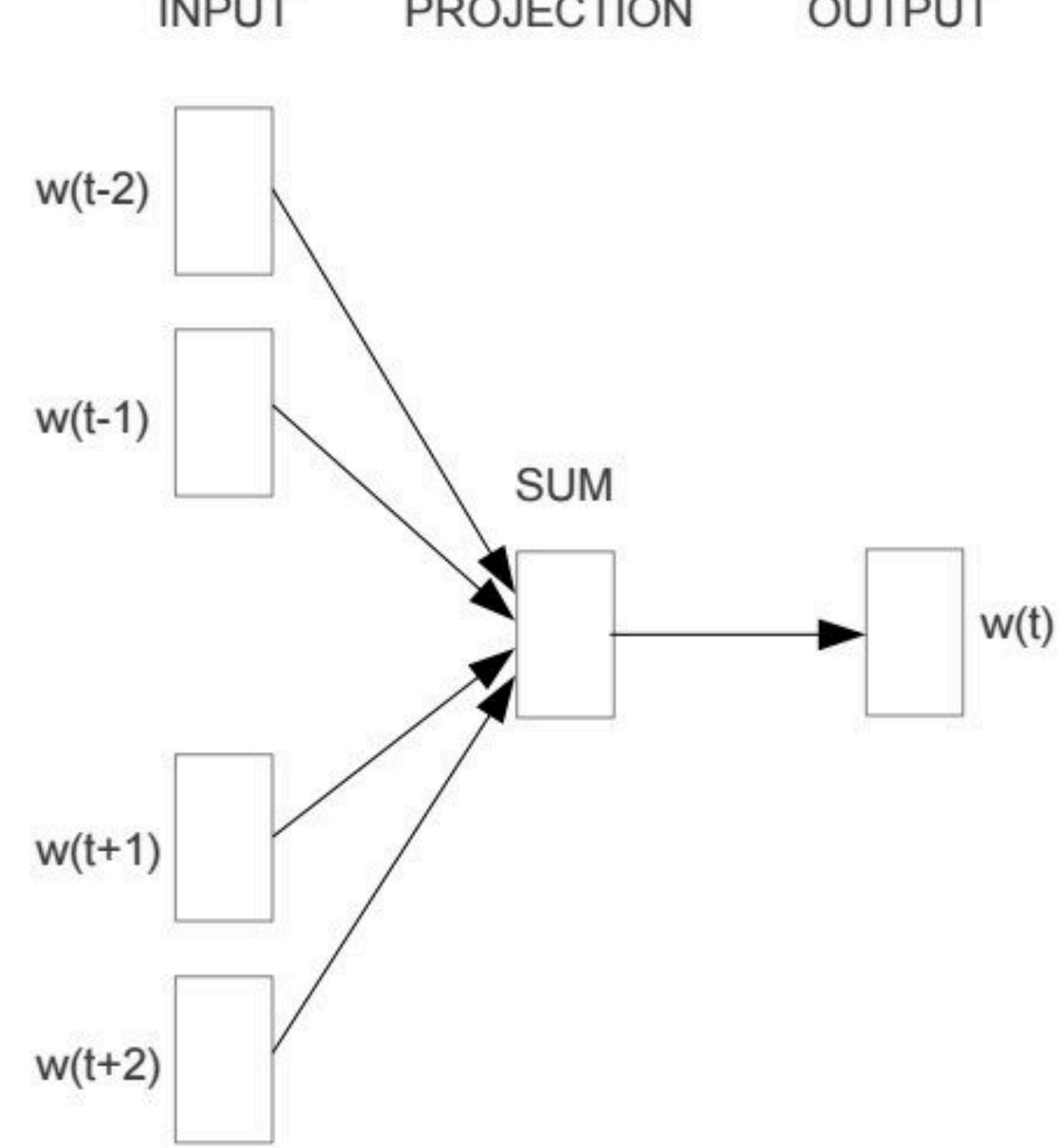
$$v_{\text{the}} = \begin{pmatrix} 0.234 \\ 0.266 \\ 0.239 \\ -0.199 \end{pmatrix}$$

$$v_{\text{language}} = \begin{pmatrix} 0.290 \\ -0.441 \\ 0.762 \\ 0.982 \end{pmatrix}$$

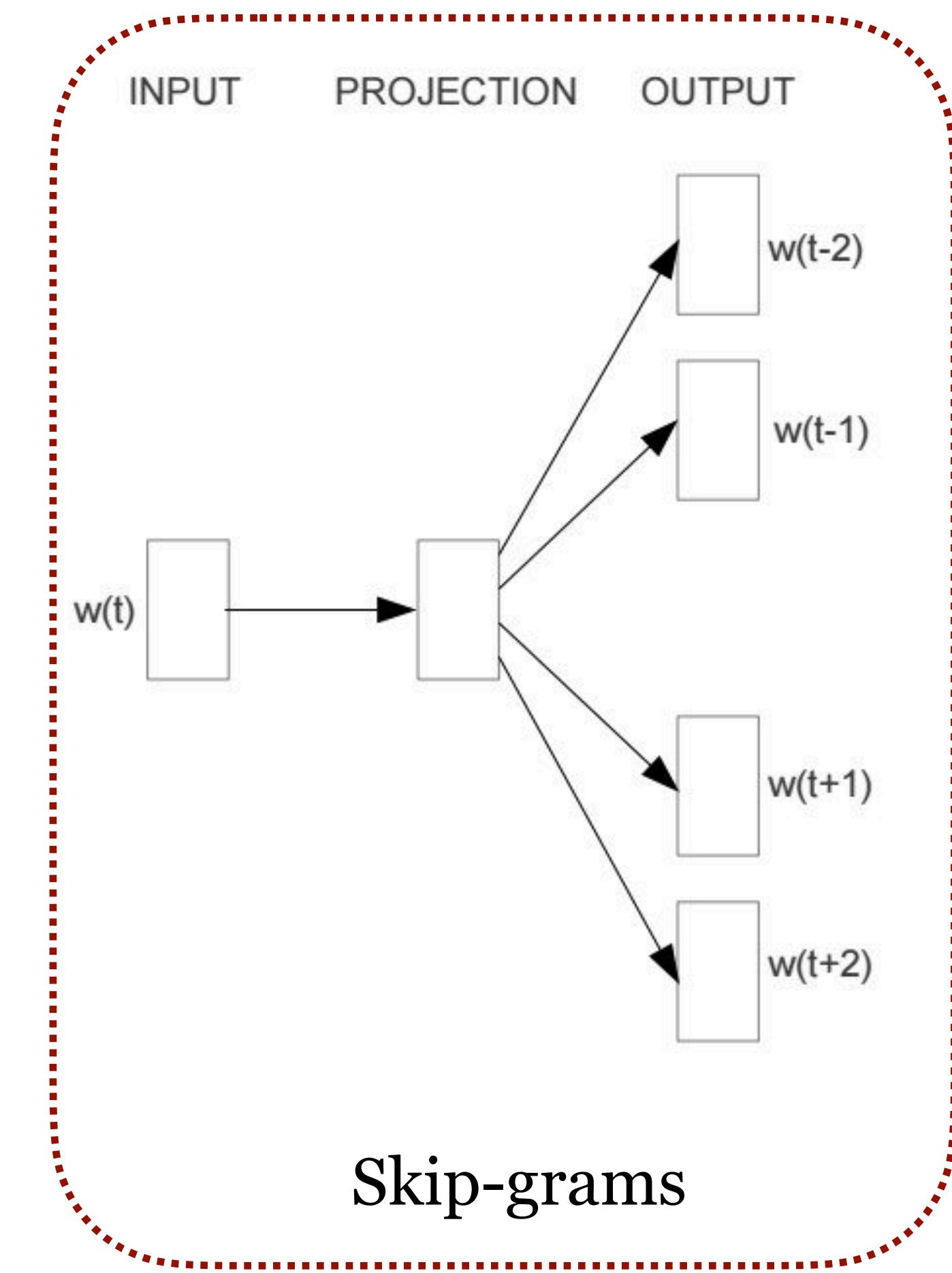
Learn f by training classifiers to predict words and take learned weights as word vectors.

Word2vec

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



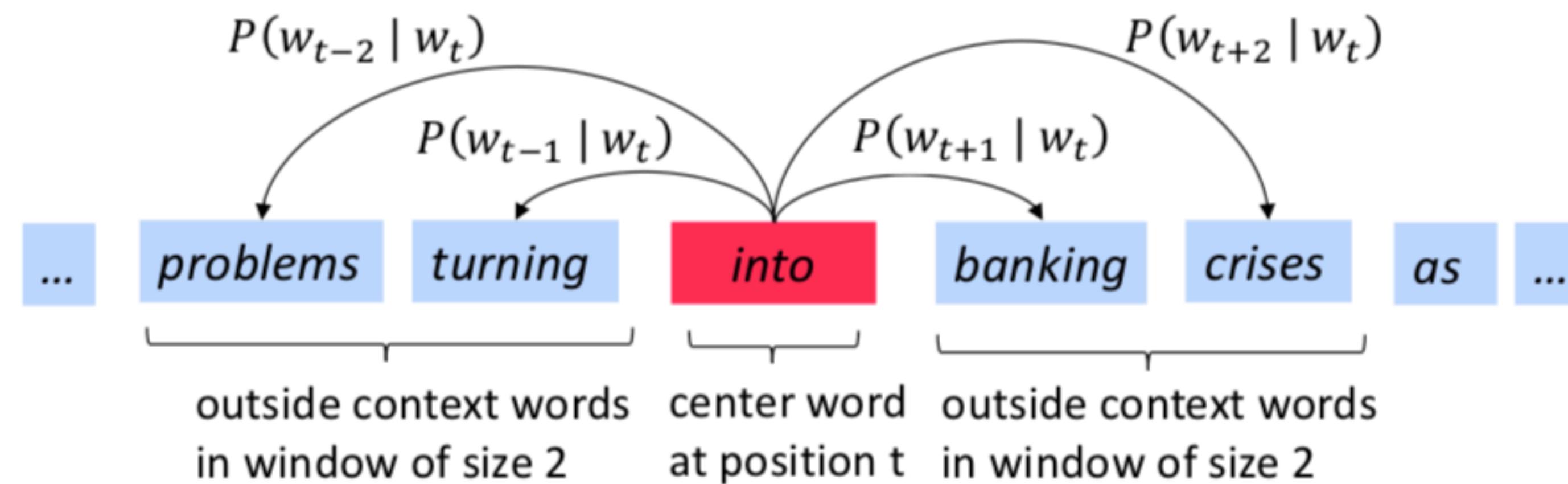
Continuous Bag of Words (CBOW)



Skip-grams

Skip-gram

- The idea: we want to use words to **predict** their context words
- Context: a fixed window of size 2



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 is to be a context word

$$P(c | t)$$

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

Goal

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

(apricot, **jam**)

(**apricot**, aardvark)

Return probability that c is a real context word:

$$P(c | t)$$

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 

- 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 $P(w_{t+j} | w_t; \theta)$?

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

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

$\mathbf{v}_{i'} \in \mathbb{R}^d$: embedding for context word i'

- Use inner product $\mathbf{u}_i \cdot \mathbf{v}_{i'}$ to measure how likely word i appears with context word i' , the larger the better

“softmax” we learned last time!

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

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

Calculating all the gradients together!

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

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

Q: How many parameters are in total?

We can apply stochastic gradient descent (SGD)!

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

Let's walk through the math..

Warm-up

$$f(x) = \exp(x)$$

$$\frac{df}{dx} = \exp(x)$$

$$f(x) = \log(x)$$

$$\frac{df}{dx} = \frac{1}{x}$$

chain rule:

$$f(x) = f_1(f_2(x))$$

$$\frac{df}{dx} = \frac{df_1(z)}{dz} \frac{df_2(x)}{dx} \quad z = f_2(x)$$

$$f(\mathbf{x}) = \mathbf{x} \cdot \mathbf{a}$$

$$\frac{\partial f}{\partial \mathbf{x}} = \mathbf{a}$$

$$\frac{\partial f}{\partial \mathbf{x}} = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n} \right]$$

Computing the gradients

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

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

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

Make sure you know how to do this!

Putting it together

- Input: text corpus, context size m , embedding size d , V
- Initialize $\mathbf{u}_i, \mathbf{v}_i$ 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?

Skip-gram with negative sampling (SGNS)

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

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

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

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

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

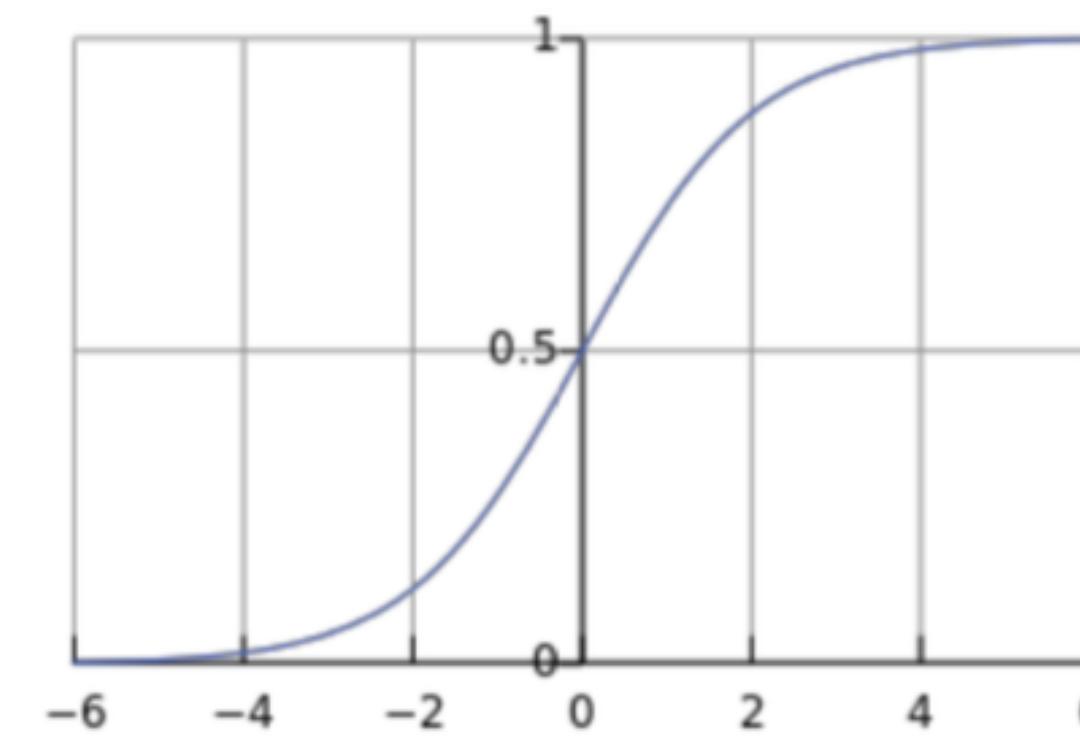
Skip-gram with negative sampling (SGNS)

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

positive examples +	
t	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

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

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



Same as training a **logistic regression** for binary classification!

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

Skip-gram with negative sampling (SGNS)

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. Use logistic regression to train a classifier to distinguish those two cases
4. Use the weights as the embeddings

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:

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

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

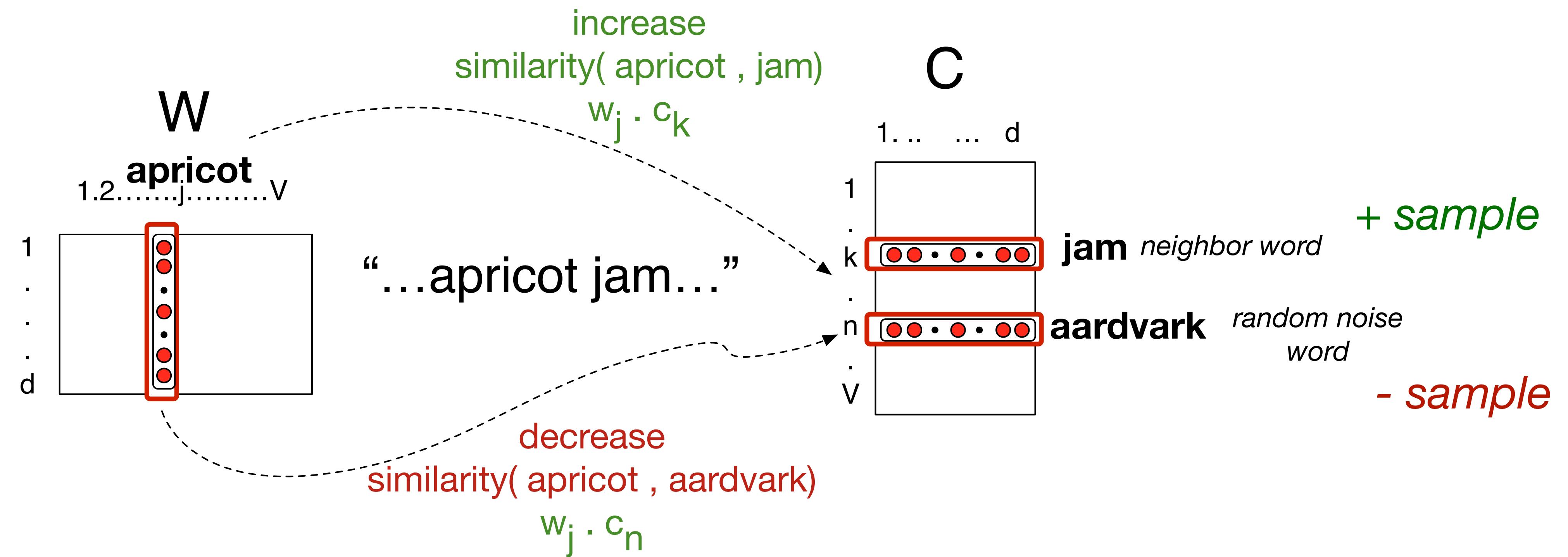
SGNS: Basic Setup

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

- 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.



SGNS: Learning the classifier

Iterative process.

We'll start with 0 or random weights

Then adjust the word weights to

- make the positive pairs more likely
- and the negative pairs less likely

over the entire training set

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 $P_\alpha(w)$

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

$\alpha = \frac{3}{4}$ 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 :

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

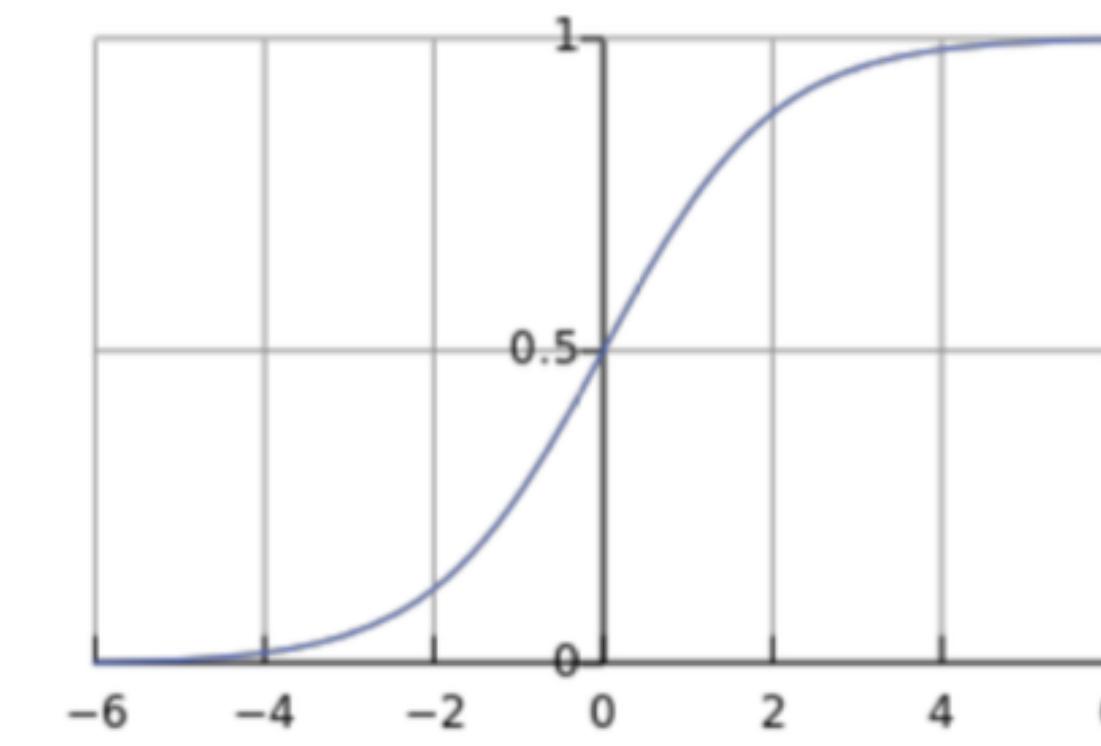
Skip-gram with negative sampling (SGNS)

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

positive examples +	
t	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

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

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

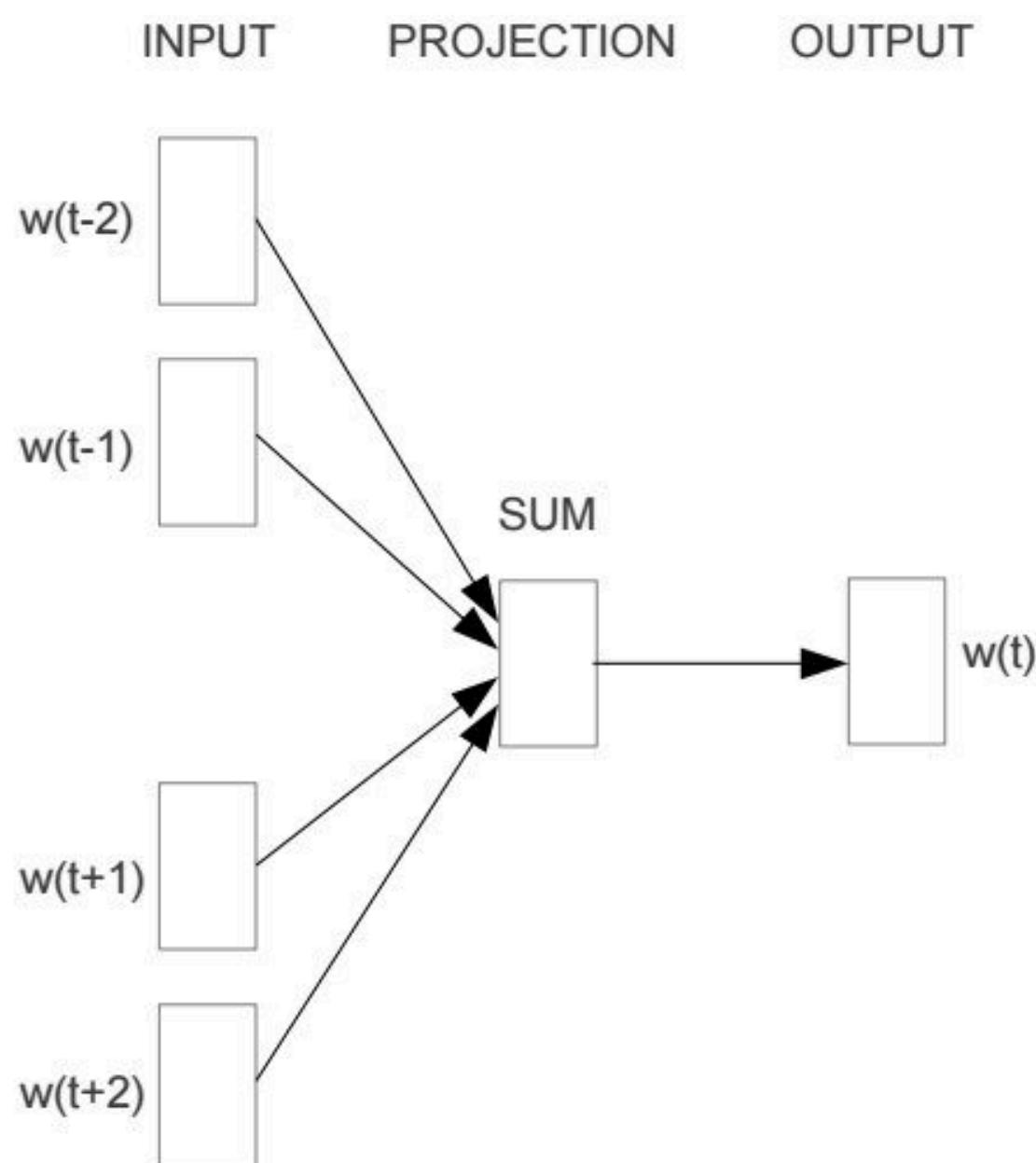


Same as training a **logistic regression** for binary classification!

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

(see also <http://jalammar.github.io/illustrated-word2vec/>)

Continuous Bag of Words (CBOW)



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

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

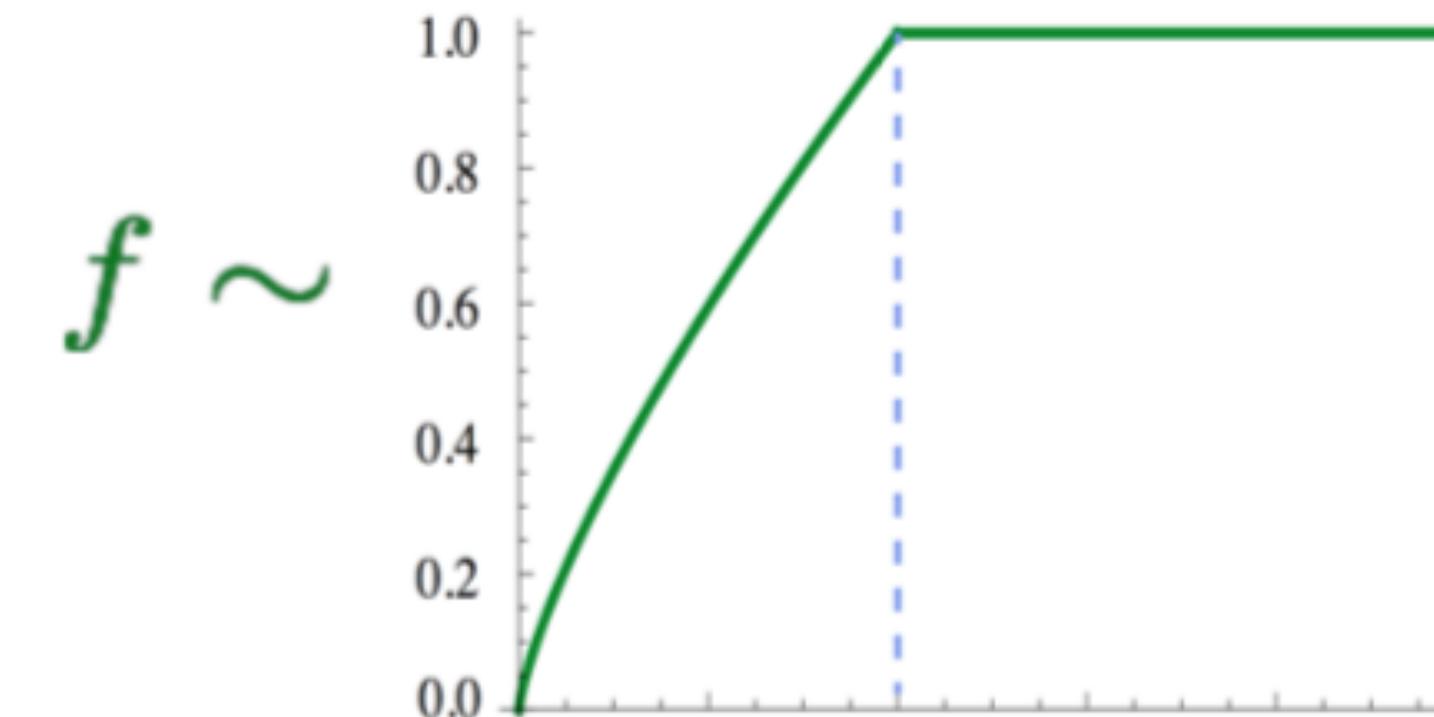
$$P(w_t | \{w_{t+j}\}) = \frac{\exp(\mathbf{u}_{w_t} \cdot \bar{\mathbf{v}}_t)}{\sum_{k \in V} \exp(\mathbf{u}_k \cdot \bar{\mathbf{v}}_t)}$$

GloVe: Global Vectors

- Let's take the global co-occurrence statistics: $X_{i,j}$

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

- Training faster
- Scalable to very large corpora



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

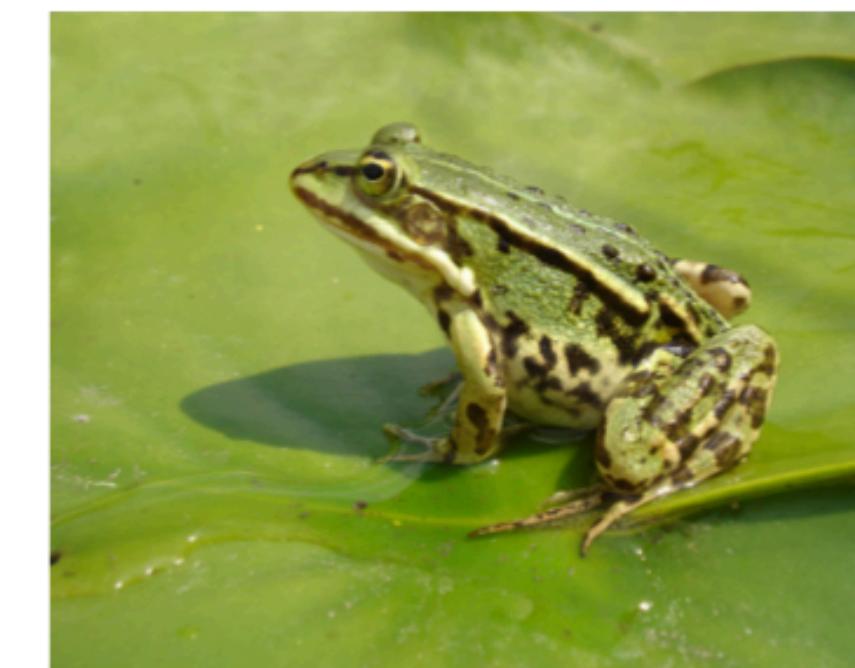
GloVe: Global Vectors

Nearest words to
[frog](#):

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



litoria



rana



leptodactylidae



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>

- 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 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

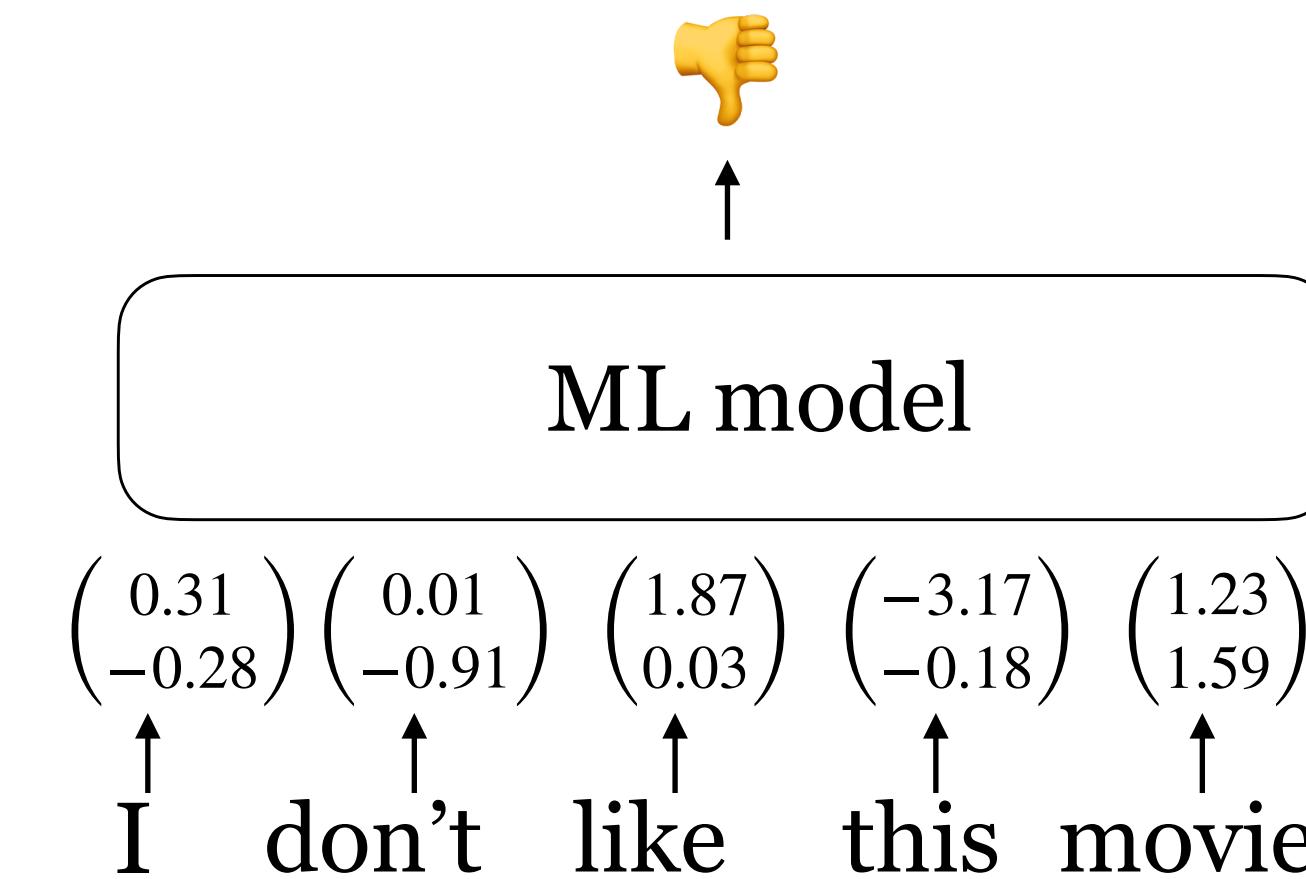
Differ in algorithms, text corpora, dimensions, cased/uncased...

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

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 [†]	6B	57.2	65.6	68.2	57.0	32.5
SG [†]	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	75.9	83.6	82.9	59.6	47.8
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

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

syntactic

Chicago:Illinois \approx Philadelphia: ? bad:worst \approx cool: ?

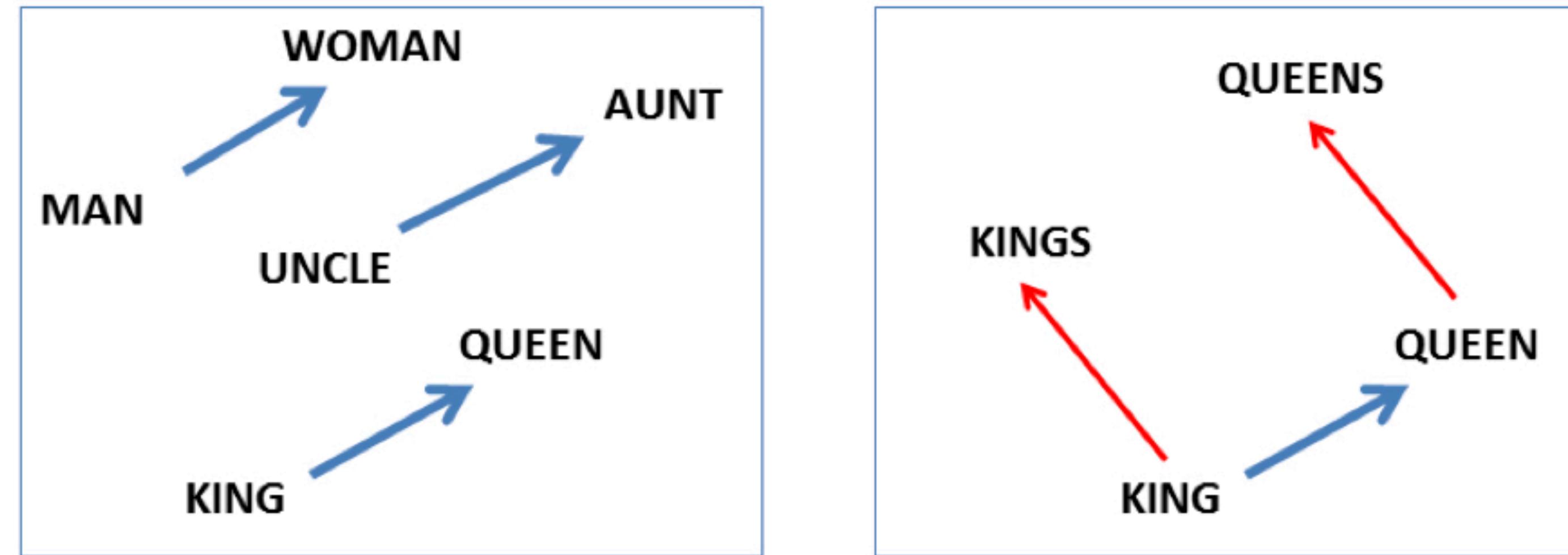
More examples at

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

Analogy: Embeddings capture relational meaning!

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

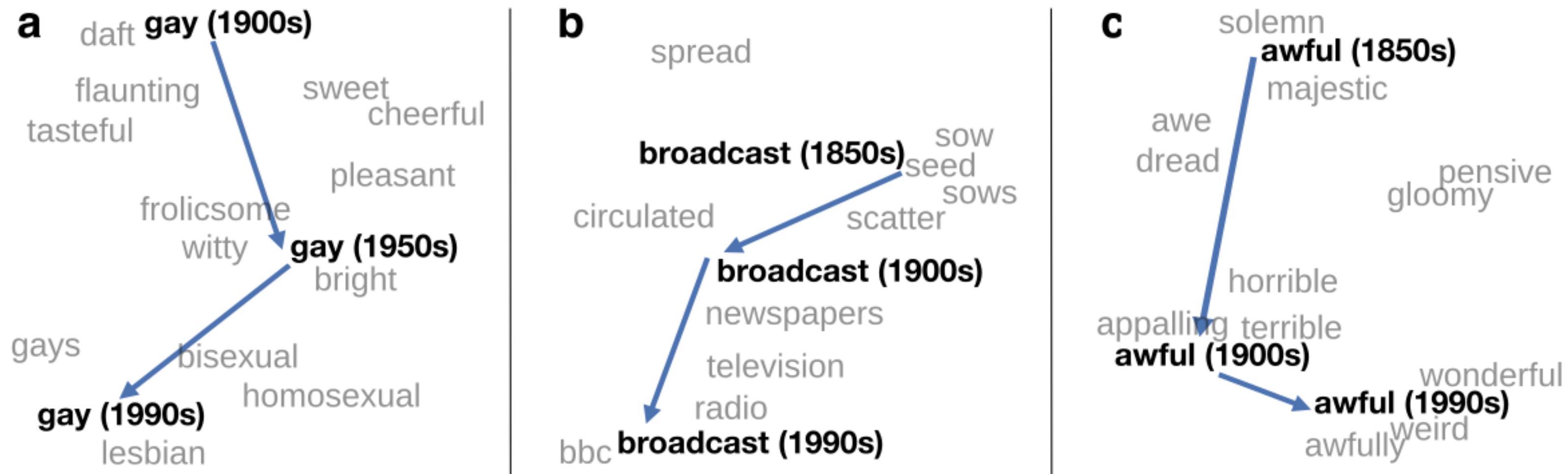
$\text{vector('Paris')} - \text{vector('France')} + \text{vector('Italy')} \approx \text{vector('Rome')}$



Applications of Word Embeddings

Embeddings can help study word history!

Project 300 dimensions down into 2

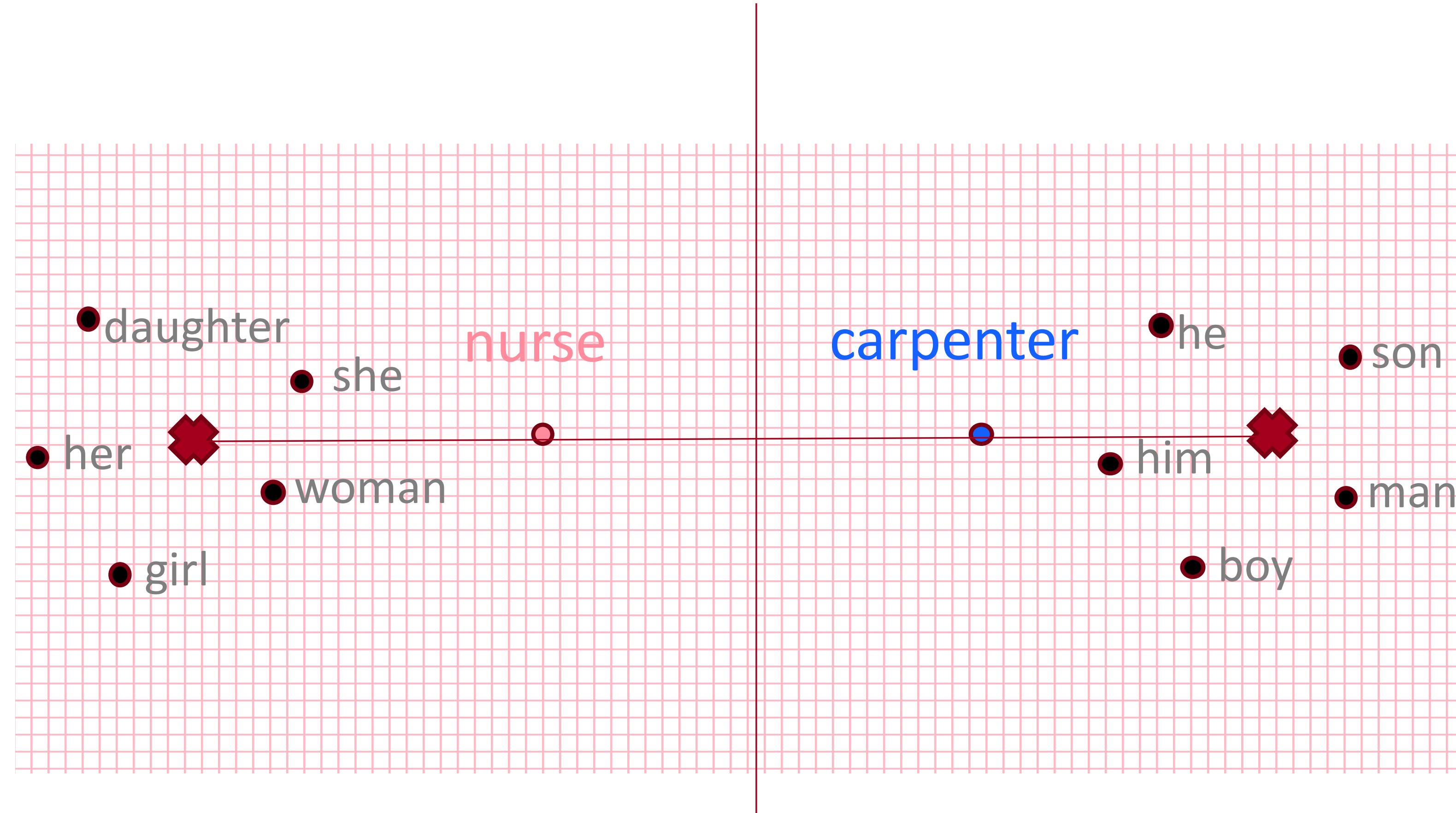


Will Hamilton

Train embeddings on old books to study changes in word meaning

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

Bias in word embeddings



Embeddings reflect cultural bias
Biases change over time

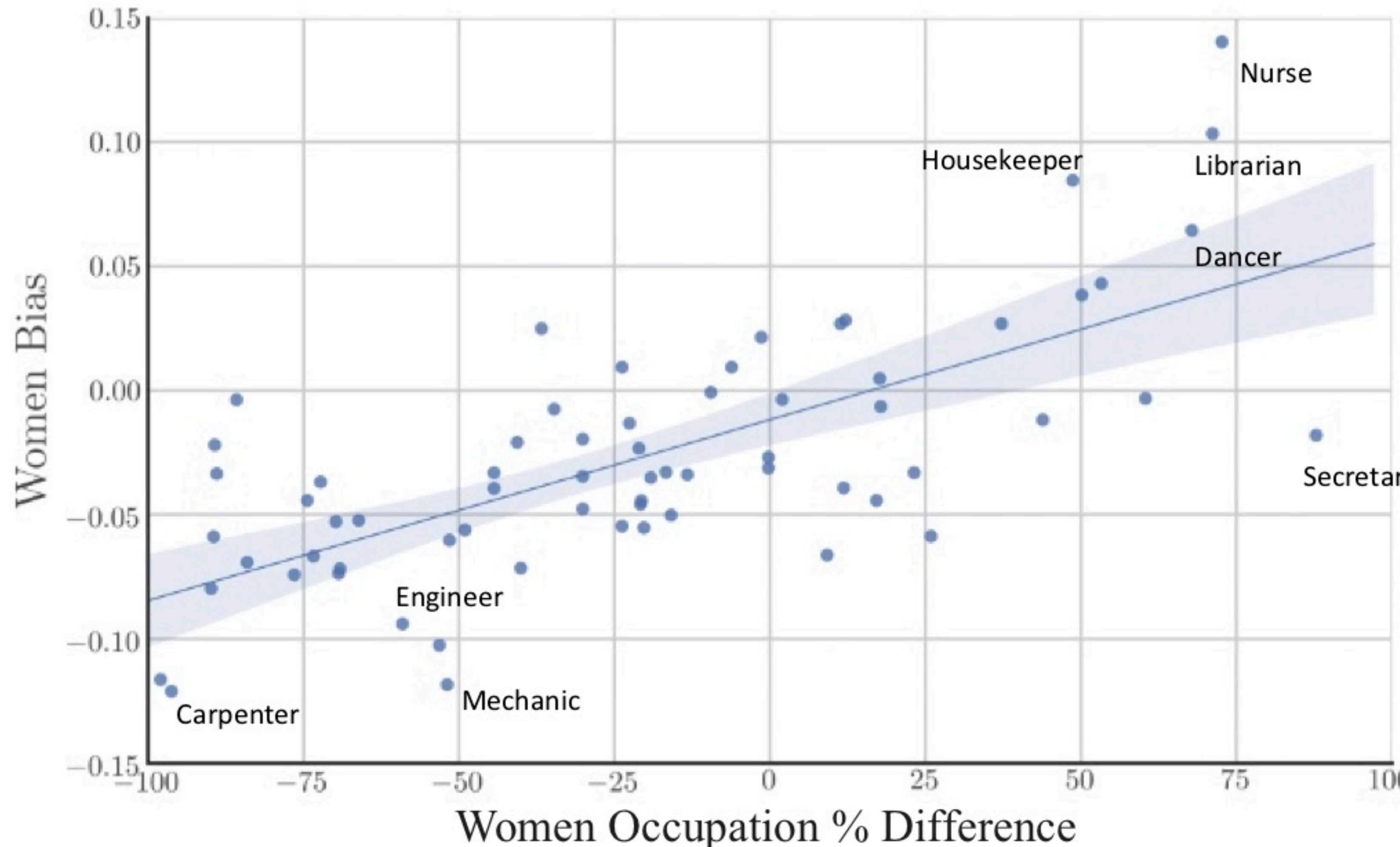
Using word embeddings to study bias

From the historical embeddings for each decade

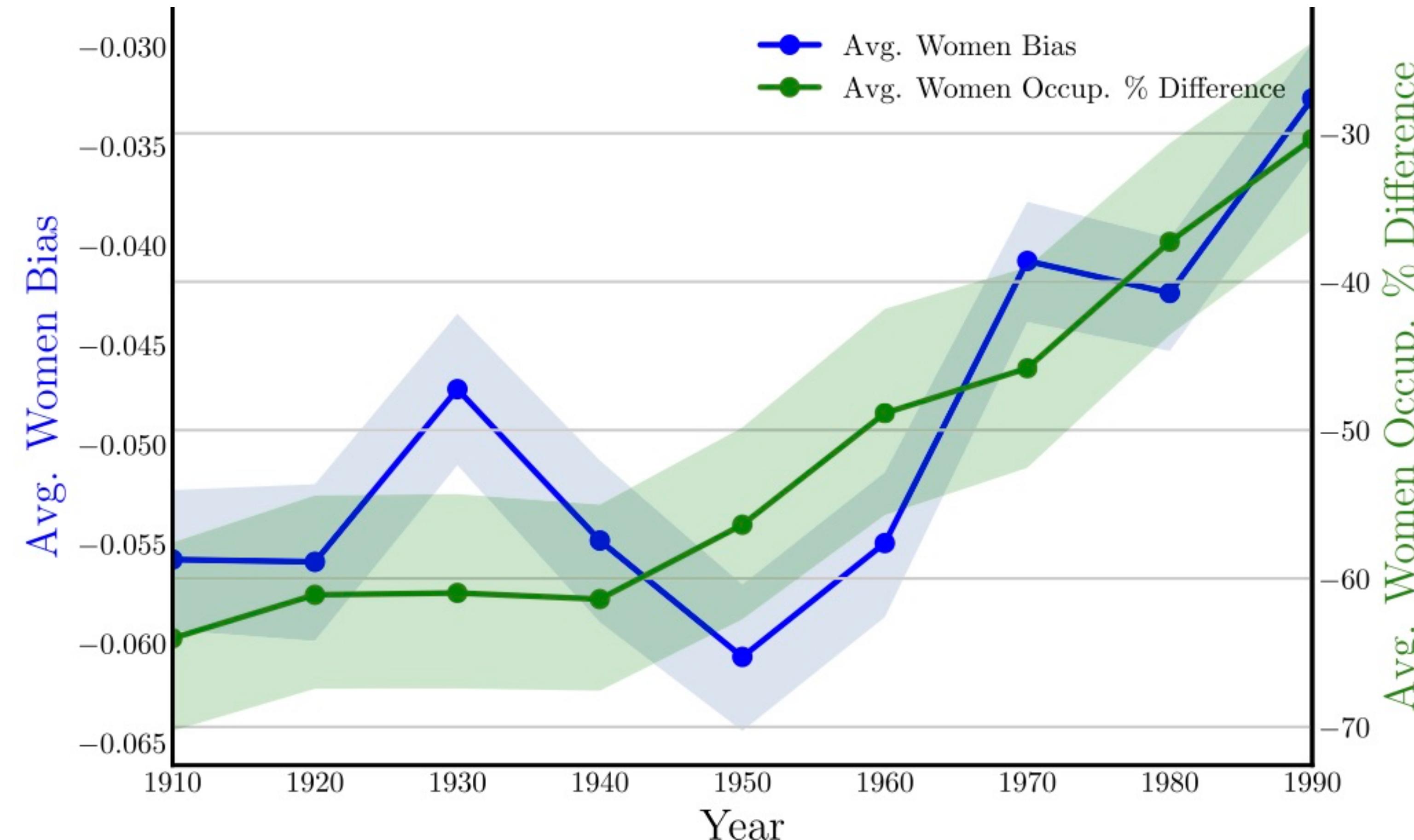
1. Compute historical biases of each word:
 - Gender bias: how much closer a word is to "woman" synonyms than "man" synonyms.
 - Chinese bias: how much closer a word is to Chinese names (Wang, Yang, Chen) than Anglo names
2. Correlate with occupational data from historical census
3. Look at how all these change over time

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

Embedding bias correlates with actual occupation data



Embeddings reflects gender bias in occupations across time (1910-1990)



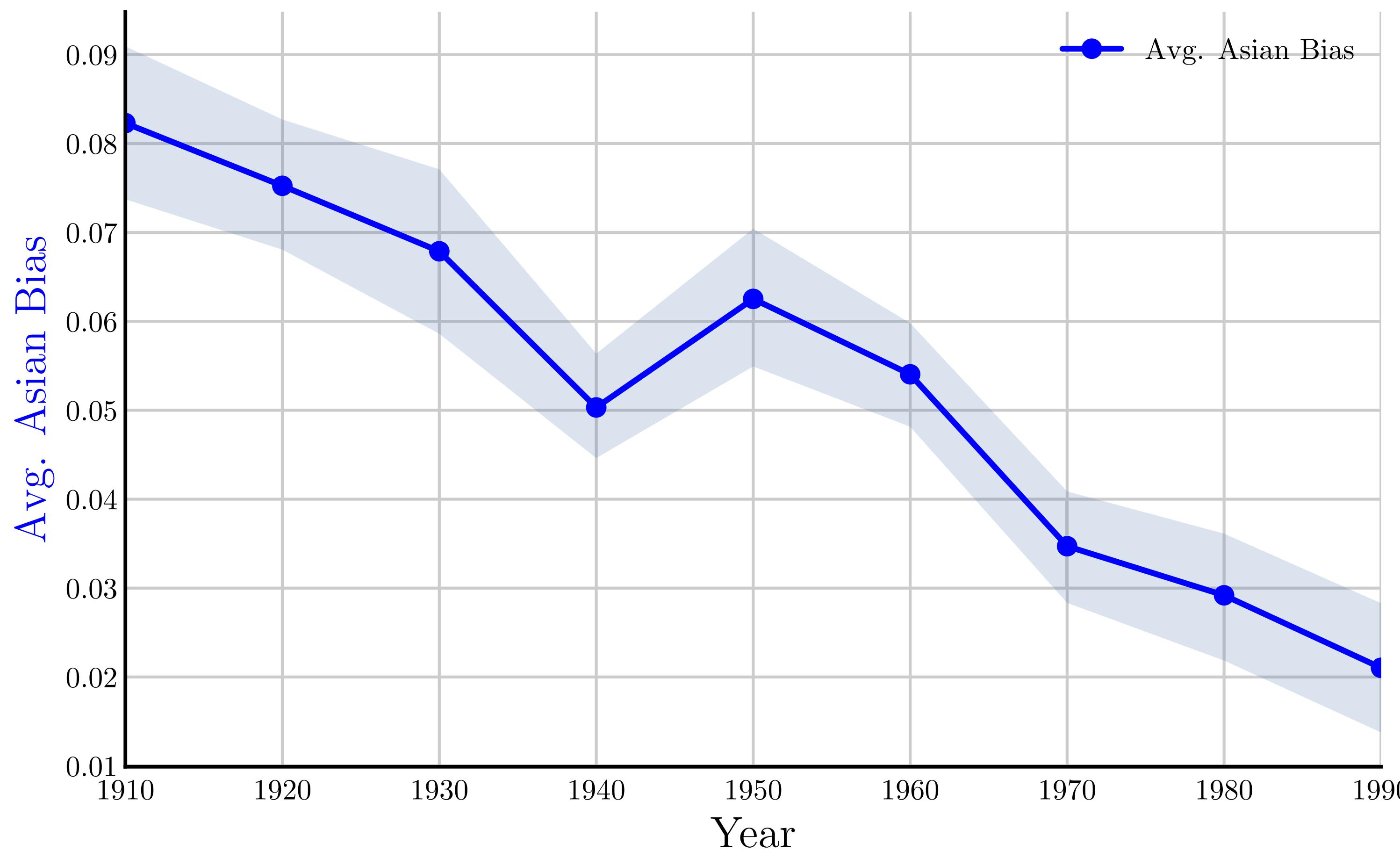
Embeddings reflect ethnic stereotypes over time

- Princeton trilogy experiments
- Attitudes toward ethnic groups (1933, 1951, 1969)
scores for adjectives
 - *industrious, superstitious, nationalistic*, etc
- Cosine of Chinese name embeddings with those
adjective embeddings correlates with human ratings.

Cultural bias over time

1910

Irresponsible
Envious
Barbaric
Aggressive
Transparent
Monstrous
Hateful
Cruel
Greedy
Bizarre



Cultural bias over time

1910	1950	1990
Irresponsible	Disorganized	Inhibited
Envious	Outrageous	Passive
Barbaric	Pompous	Dissolute
Aggressive	Unstable	Haughty
Transparent	Effeminate	Complacent
Monstrous	Unprincipled	Forceful
Hateful	Venomous	Fixed
Cruel	Disobedient	Active
Greedy	Predatory	Sensitive
Bizarre	Boisterous	Hearty

Beyond Simple Word Embeddings

pike

- A sharp point or staff
 - A type of elongated fish
 - A railroad line or system
 - A type of road
 - The future (coming down the pike)
 - A type of body position (as in diving)
 - To kill or pierce with a pike
 - To make one's way (pike along)
 - In Australian English, pike means to pull out from doing something: *I reckon he could have climbed that cliff, but he piked!*

Sense Embeddings

