

# Word Embeddings

Wei Xu

(many slides from Greg Durrett)

# This Lecture

---

- ▶ Word representations
- ▶ word2vec/GloVe
- ▶ Evaluating word embeddings

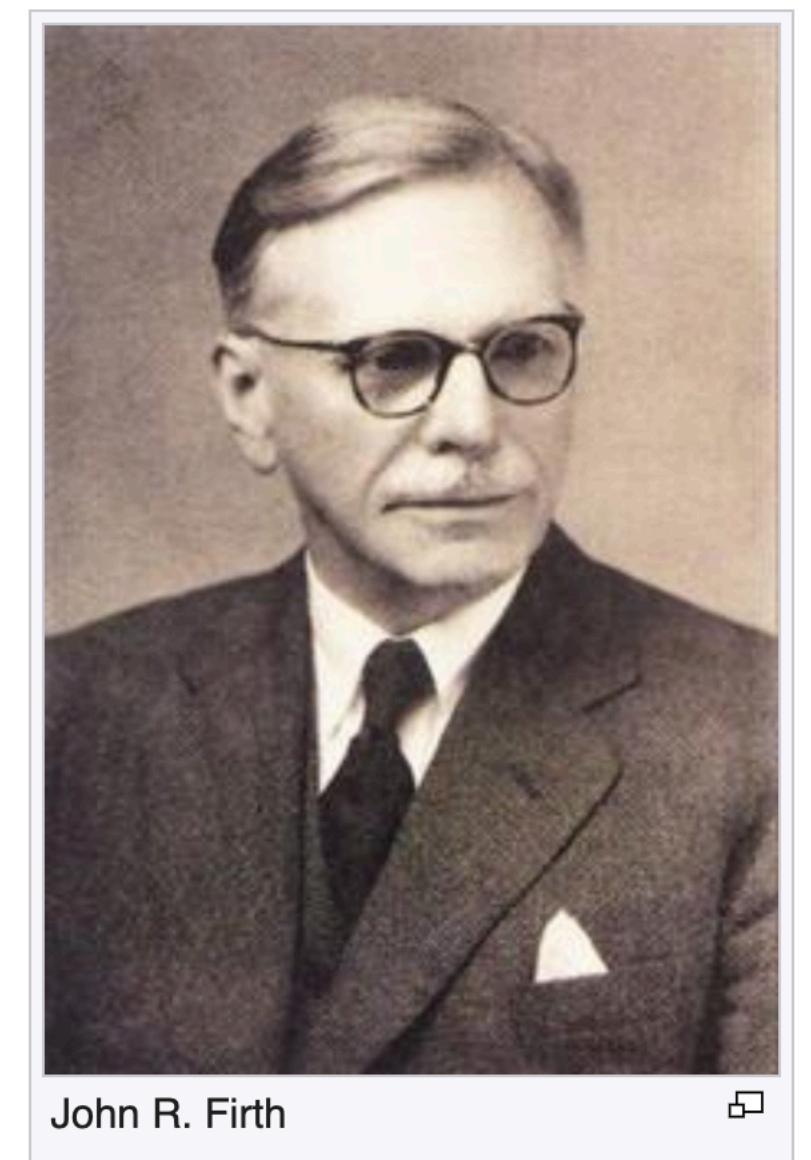
# Word Representations

# Word Representations

---

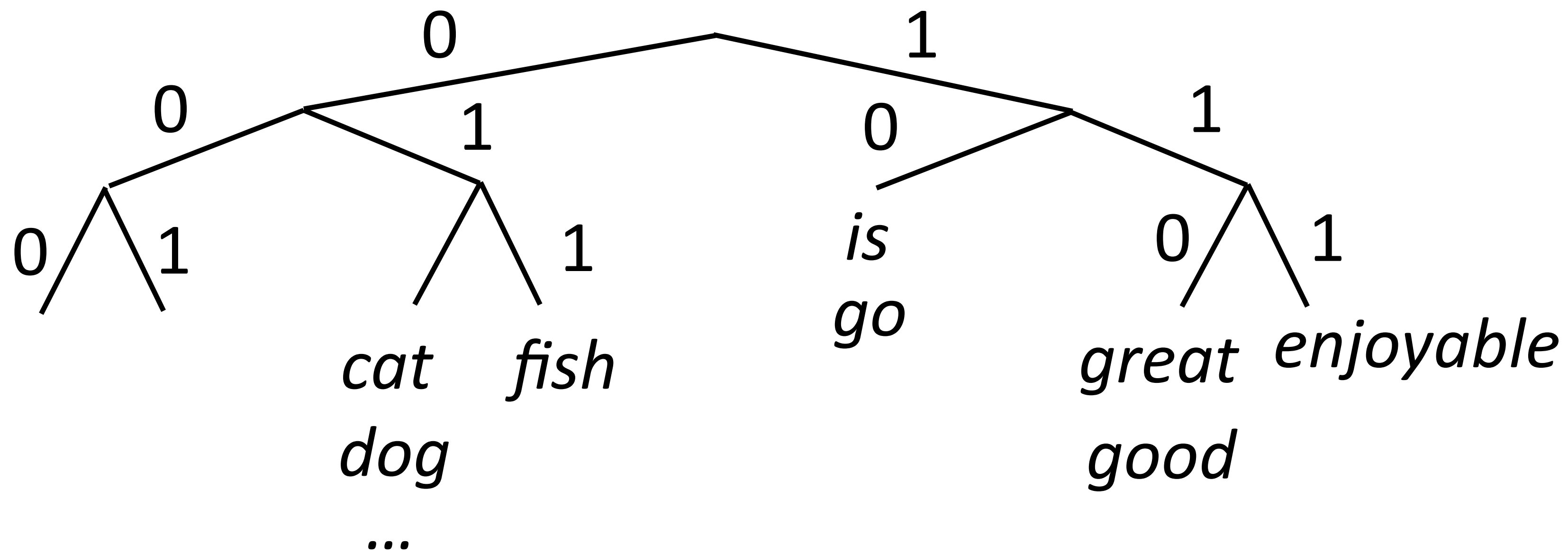
- ▶ Neural networks work very well at continuous data, but words are discrete
- ▶ Continuous model  $\leftrightarrow$  expects continuous semantics from input
- ▶ “You shall know a word by the company it keeps” Firth (1957)

A bottle of ***tesgüino*** is on the table  
Everybody likes ***tesgüino***  
***Tesgüino*** makes you drunk  
We make ***tesgüino*** out of corn.



# Discrete Word Representations

- ▶ Brown clusters: hierarchical agglomerative *hard* clustering (each word has one cluster, not some posterior distribution like in mixture models)



- ▶ Maximize  $P(w_i|w_{i-1}) = P(c_i|c_{i-1})P(w_i|c_i)$
  - ▶ Useful features for tasks like NER, not suitable for NNs
- Brown et al. (1992)

# Discrete Word Representations

- ▶ Brown clusters: hierarchical agglomerative *hard* clustering
- ▶ We give a very brief sketch of the algorithm here:

- $k$ : a hyper-parameter, sort words by frequency
- Take the top  $k$  most frequent words, put each of them in its own cluster  $c_1, c_2, c_3, \dots, c_k$
- For  $i = (k + 1) \dots |V|$ 
  - Create a new cluster  $c_{k+1}$  (we have  $k + 1$  clusters)
  - Choose two clusters from  $k + 1$  clusters based on  $\text{quality}(C)$  and merge (back to  $k$  clusters)

$$\text{Quality}(C) = \sum_i^n \log e(w_i | C(w_i)) q(C(w_i) | C(w_{i-1})) = \sum_{c=1}^k \sum_{c'=1}^k p(c, c') \log \frac{p(c, c')}{p(c)p(c')} + G$$

mutual information  
between adjacent clusters

entropy of  
the word distribution

- Carry out  $k - 1$  final merges (*full hierarchy*)
- Running time  $O(|V|k^2 + n)$ ,  $n = \text{\#words in corpus}$



# Discrete Word Representations

- ▶ Brown clusters: hierarchical agglomerative *hard* clustering
- ▶ Example Clusters from Miller et al. 2004

mailman	10000011010111
salesman	100000110110000
bookkeeper	1000001101100010
troubleshooter	10000011011000110
bouncer	10000011011000111
technician	1000001101100100
janitor	1000001101100101
saleswoman	1000001101100110
...	
Nike	1011011100100101011100
Maytag	10110111001001010111010
Generali	10110111001001010111011
Gap	1011011100100101011110
Harley-Davidson	10110111001001010111110
Enfield	101101110010010101111110
genus	101101110010010101111111
Microsoft	10110111001001011000
Ventritex	101101110010010110010
Tractebel	1011011100100101100110
Synopsys	1011011100100101100111
WordPerfect	1011011100100101101000
....	
John	101110010000000000
Consuelo	101110010000000001
Jeffrey	101110010000000010
Kenneth	10111001000000001100
Phillip	101110010000000011010
WILLIAM	101110010000000011011
Timothy	10111001000000001110

**word cluster features** (bit string prefix)

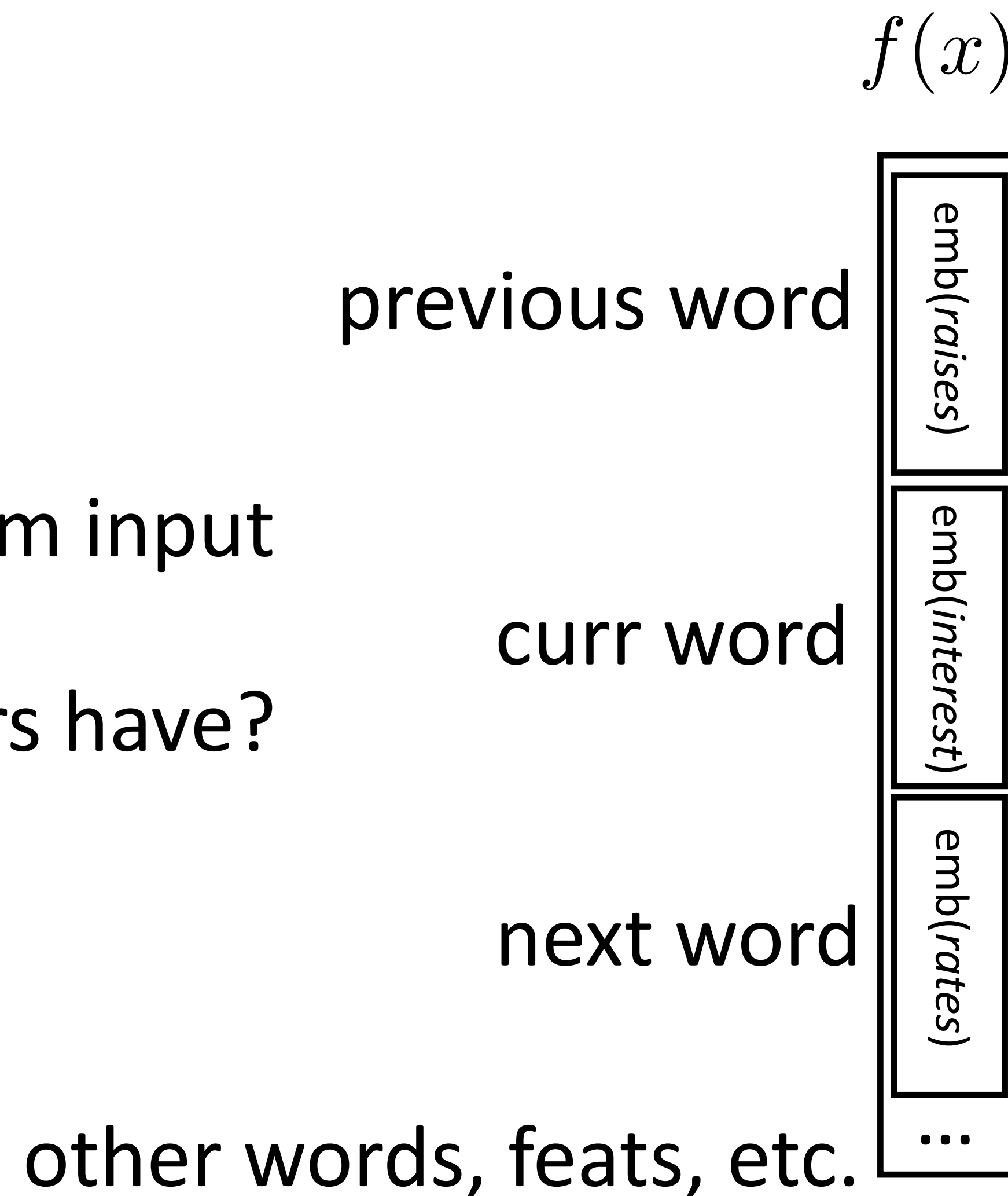
# Word Embeddings

- ▶ Part-of-speech tagging with FFNNs

??

*Fed raises **interest** rates in order to ...*

- ▶ Word embeddings for each word form input
- ▶ What properties should these vectors have?

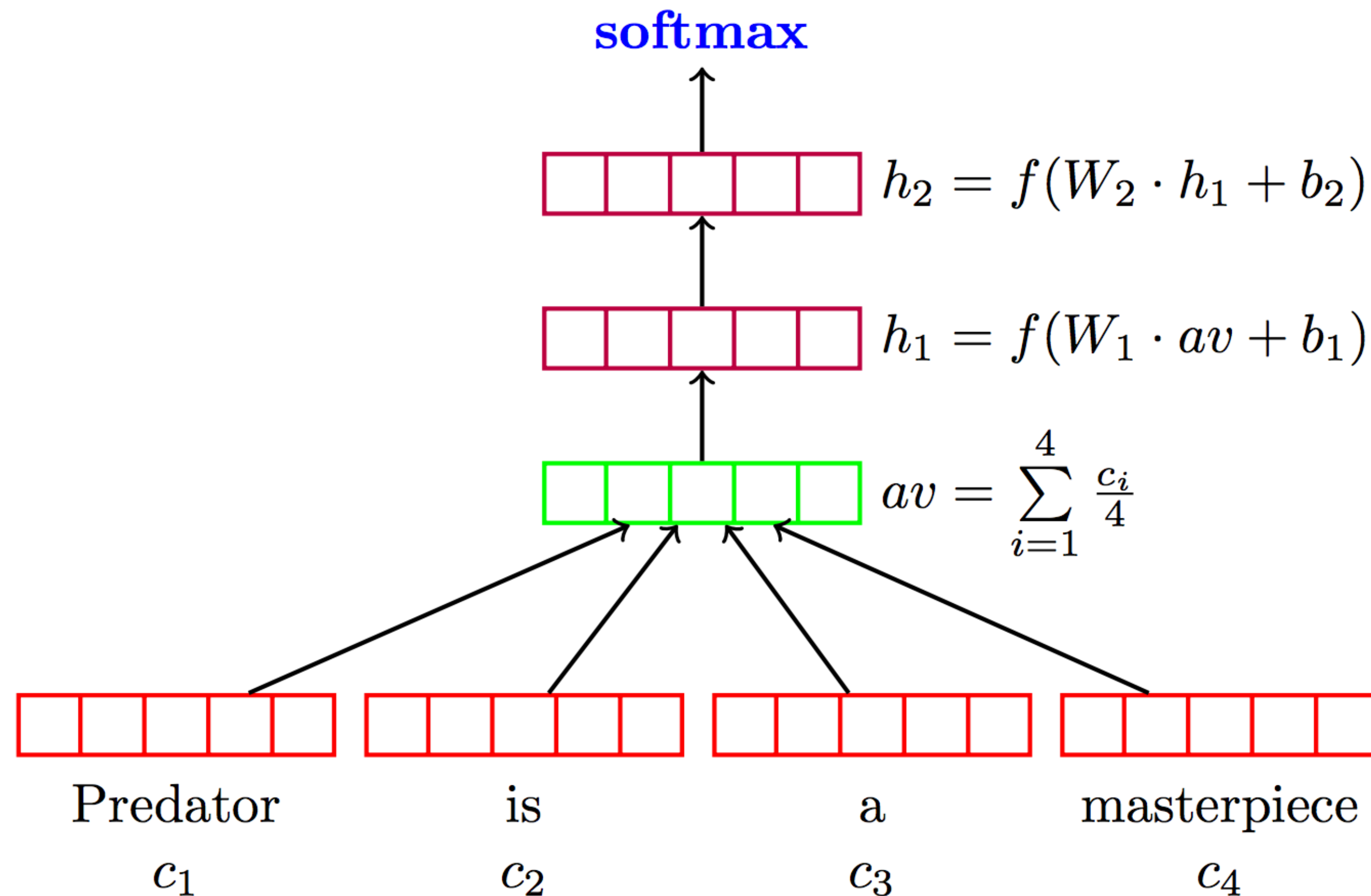


Botha et al. (2017)



# Sentiment Analysis

- ▶ Deep Averaging Networks: feedforward neural network on average of word embeddings from input



# Word Embeddings

---

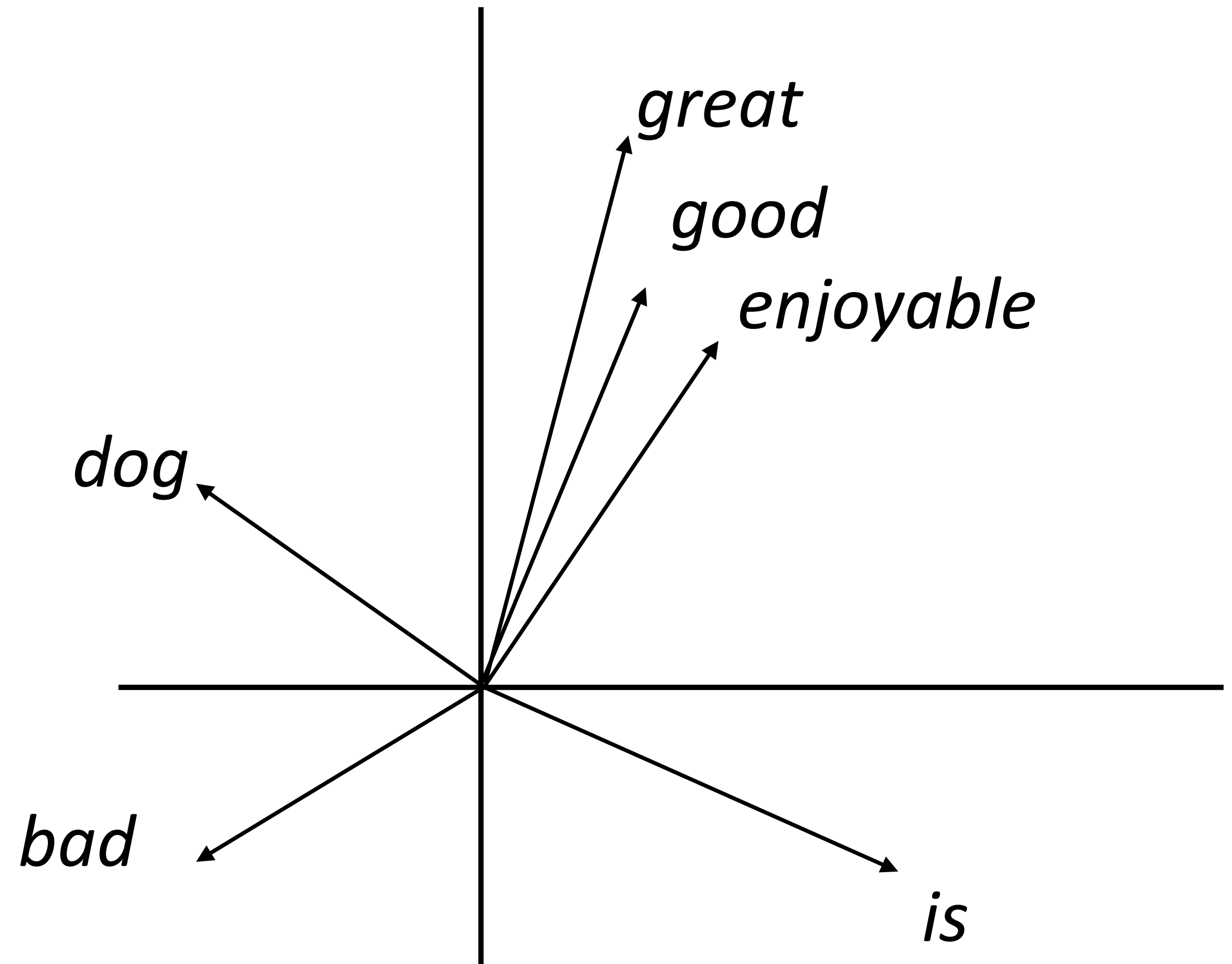
- ▶ Want a vector space where similar words have similar embeddings

*the movie was great*

$\approx$

*the movie was good*

- ▶ Goal: come up with a way to produce these embeddings
- ▶ For each word, want “medium” dimensional vector (50-300 dims) representing it.



# Word Representations

---

- ▶ Count-based:  $\text{tf} \cdot \text{idf}$ , PPMI, ...
- ▶ Class-based: Brown Clusters, ...
- ▶ Distributed prediction-based embeddings: Word2vec, GloVe, FastText, ...
- ▶ Distributed contextual embeddings: ELMo, BERT, GPT, ...
- ▶ + many more variants: multi-sense embeddings, syntactic embeddings, ...

word2vec/GloVe

# Neural Probabilistic Language Model

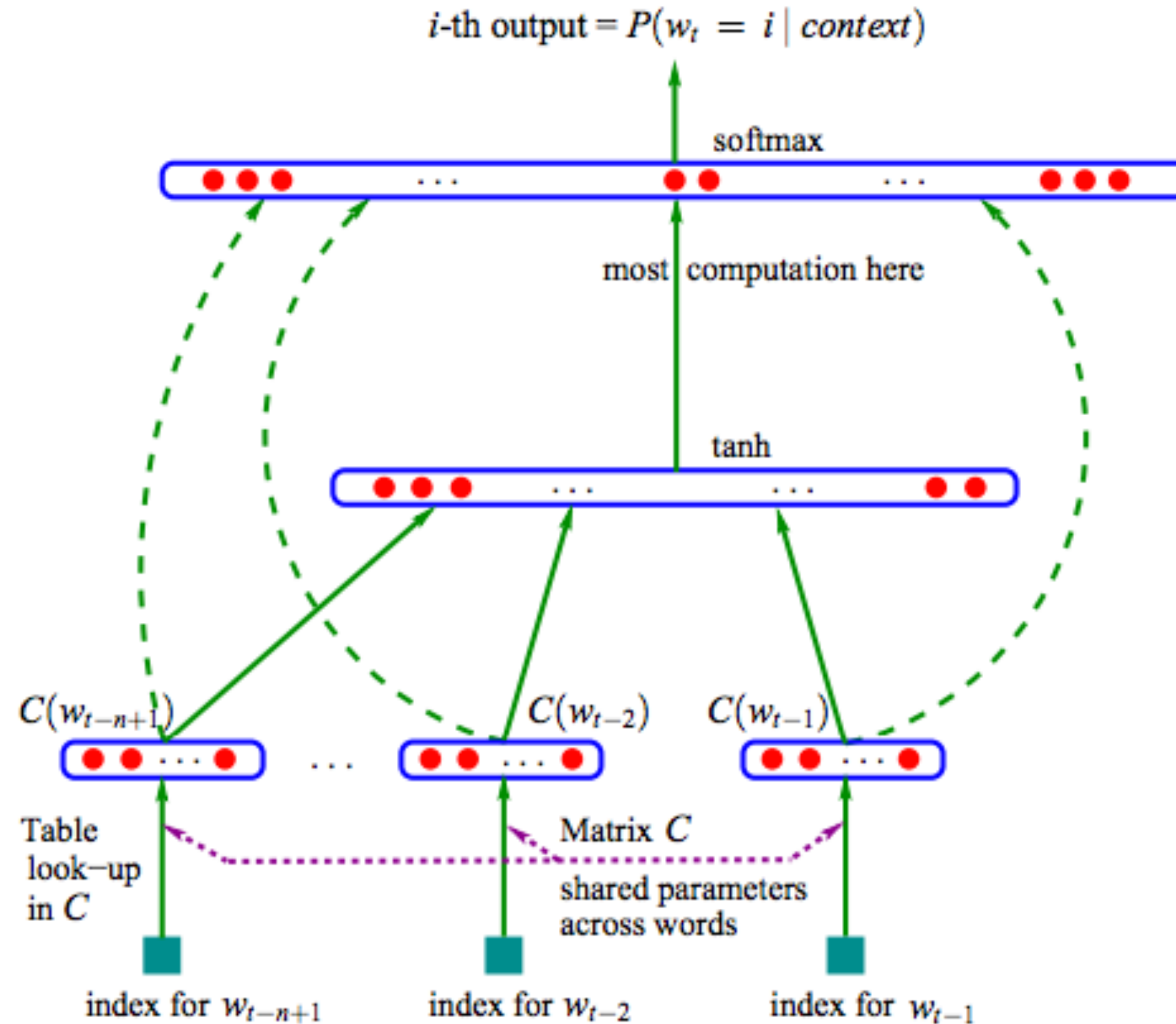


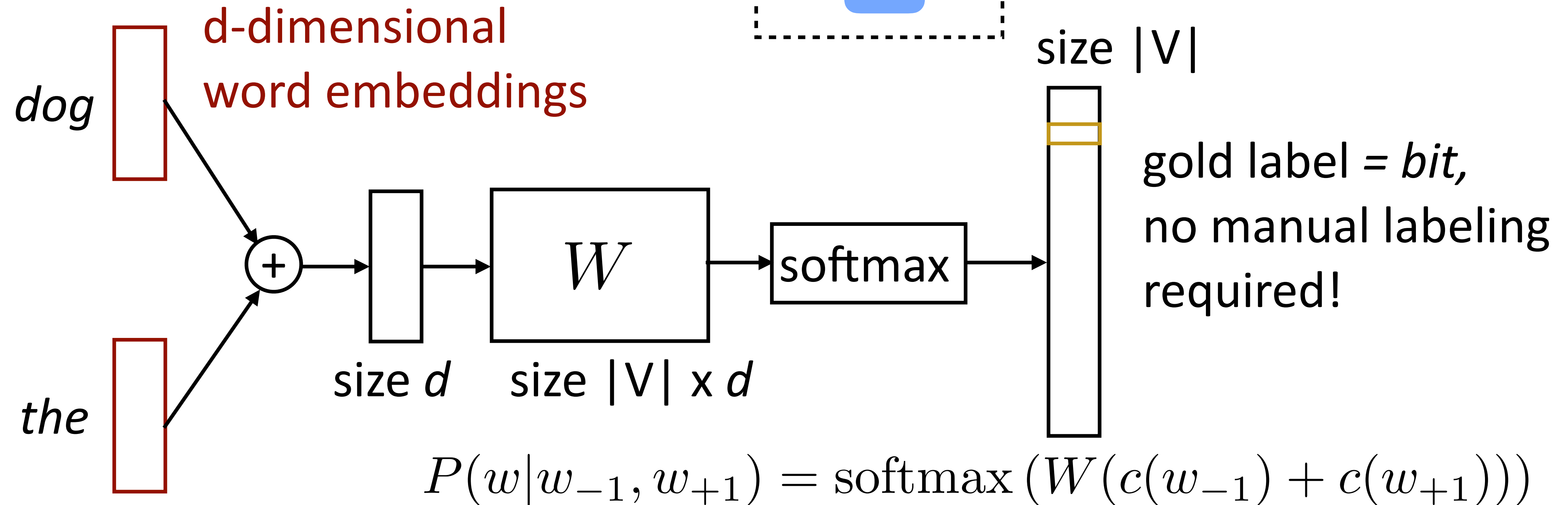
Figure 1: Neural architecture:  $f(i, w_{t-1}, \dots, w_{t-n+1}) = g(i, C(w_{t-1}), \dots, C(w_{t-n+1}))$  where  $g$  is the neural network and  $C(i)$  is the  $i$ -th word feature vector.

Bengio et al. (2003)

# word2vec: Continuous Bag-of-Words

- Predict word from context

*the dog **bit** the man*



- Parameters:  $d \times |V|$  (one  $d$ -length **context vector per voc word**),  
 $|V| \times d$  output parameters ( $W$ )

Mikolov et al. (2013)

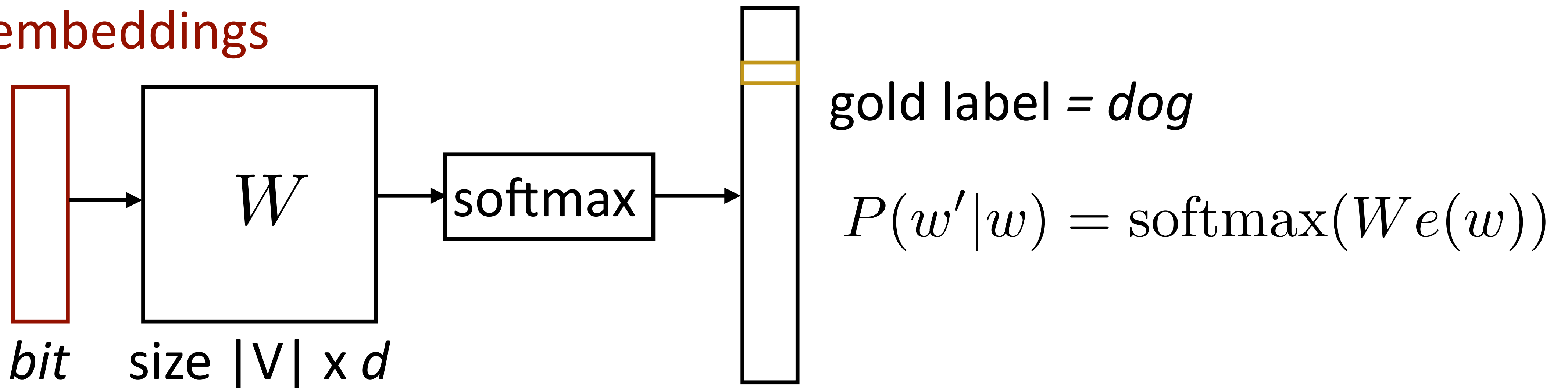


# word2vec: Skip-Gram

- Predict one word of context from word

d-dimensional  
word embeddings

*the dog bit the man*

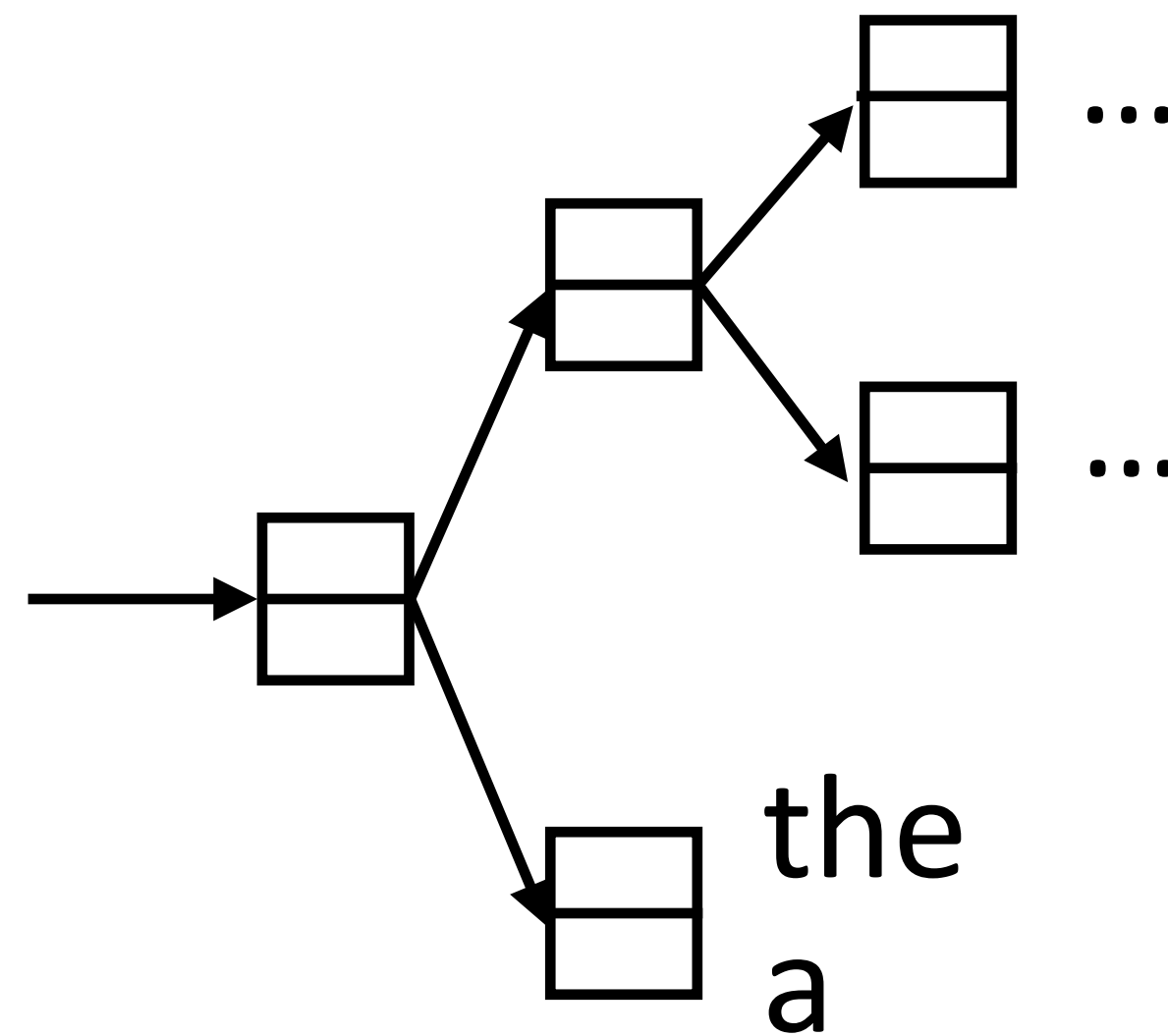
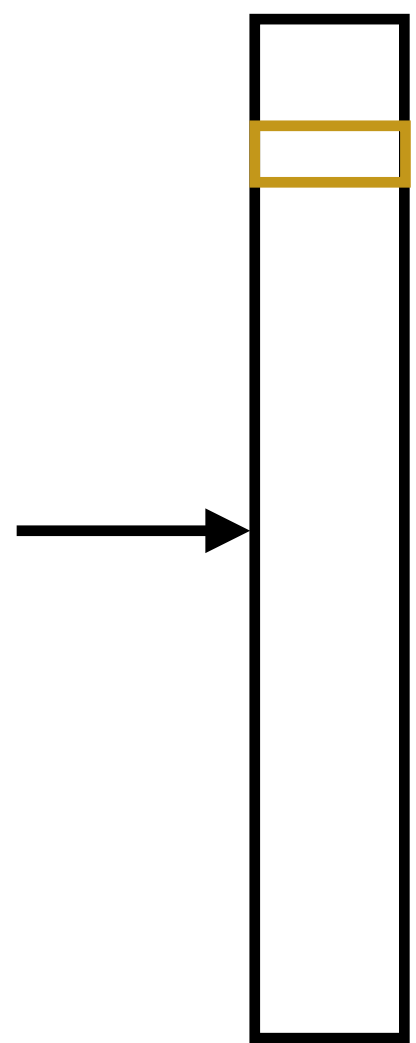


- Another training example: *bit*  $\rightarrow$  *the*
- Parameters:  $d \times |V|$  **vectors**,  $|V| \times d$  output parameters ( $W$ ) (also usable as vectors!)

# Hierarchical Softmax

$$P(w|w_{-1}, w_{+1}) = \text{softmax}(W(c(w_{-1}) + c(w_{+1}))) \quad P(w'|w) = \text{softmax}(We(w))$$

- ▶ Matmul + softmax over  $|V|$  is very slow to compute for CBOW and SG



- ▶ Huffman encode vocabulary, use binary classifiers to decide which branch to take
- ▶  $\log(|V|)$  binary decisions

- ▶ Standard softmax:  
 $O(|V|)$  dot products of size  $d$   
- per training instance per context word

- ▶ Hierarchical softmax:  
 $O(\log(|V|))$  dot products of size  $d$ ,  
 $|V| \times d$  parameters

Mikolov et al. (2013)

# Skip-Gram with Negative Sampling

- Take (word, context) pairs and classify them as “real” or not. Create random negative examples by sampling from unigram distribution

*(bit, the) => +1*

*(bit, cat) => -1*

*(bit, a) => -1*

*(bit, fish) => -1*

*the dog bit the man*



$$P(y = 1|w, c) = \frac{e^{w \cdot c}}{e^{w \cdot c} + 1}$$

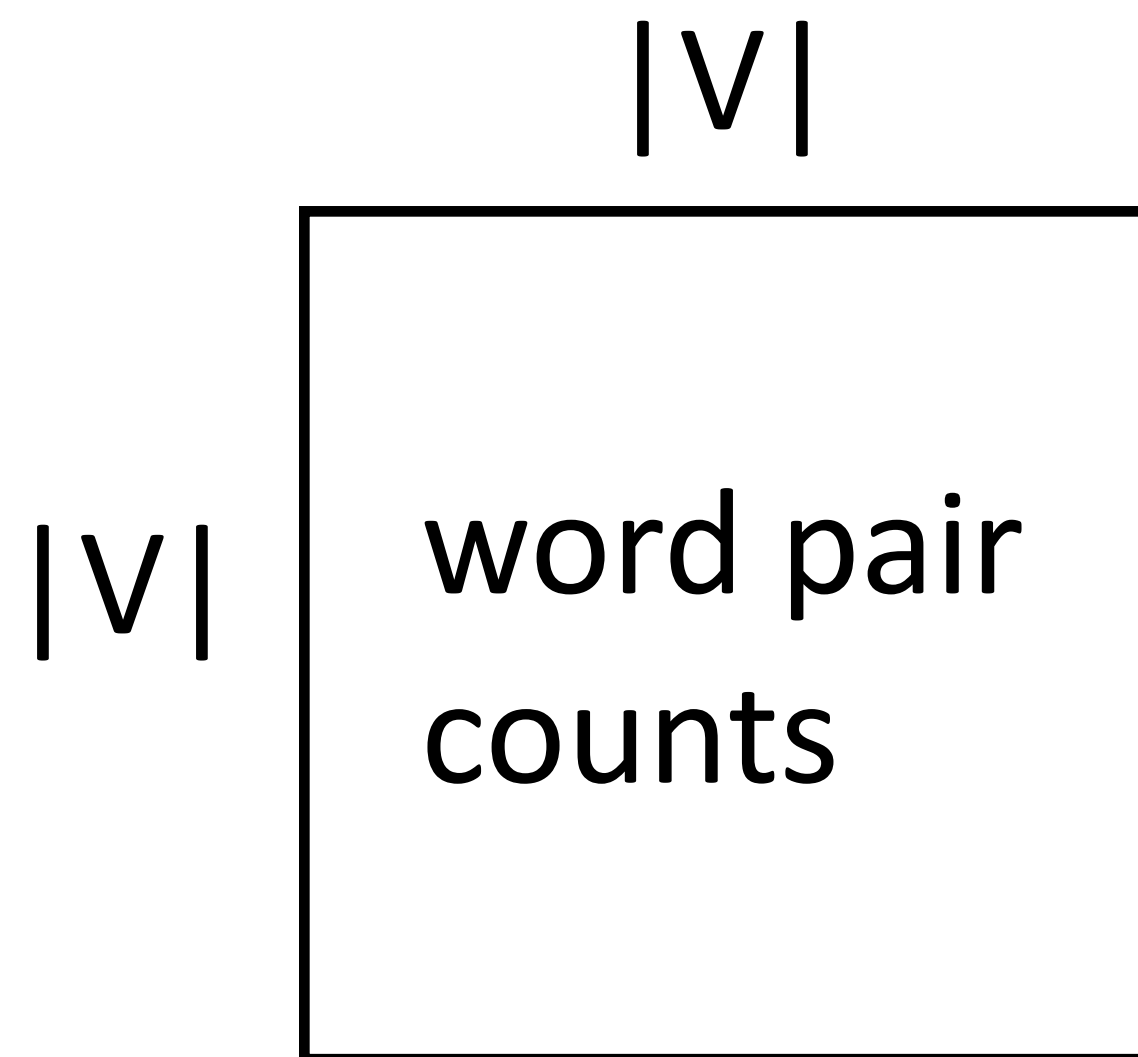
words in similar contexts select for similar  $c$  vectors

- $d \times |V|$  vectors,  $d \times |V|$  context vectors (same # of params as before)

- Objective =  $\log P(y = 1|w, c) - \sum_{i=1}^k \log P(y = 0|w_i, c)$  sampled

# Connections with Matrix Factorization

- ▶ Skip-gram model looks at word-word co-occurrences and produces two types of vectors



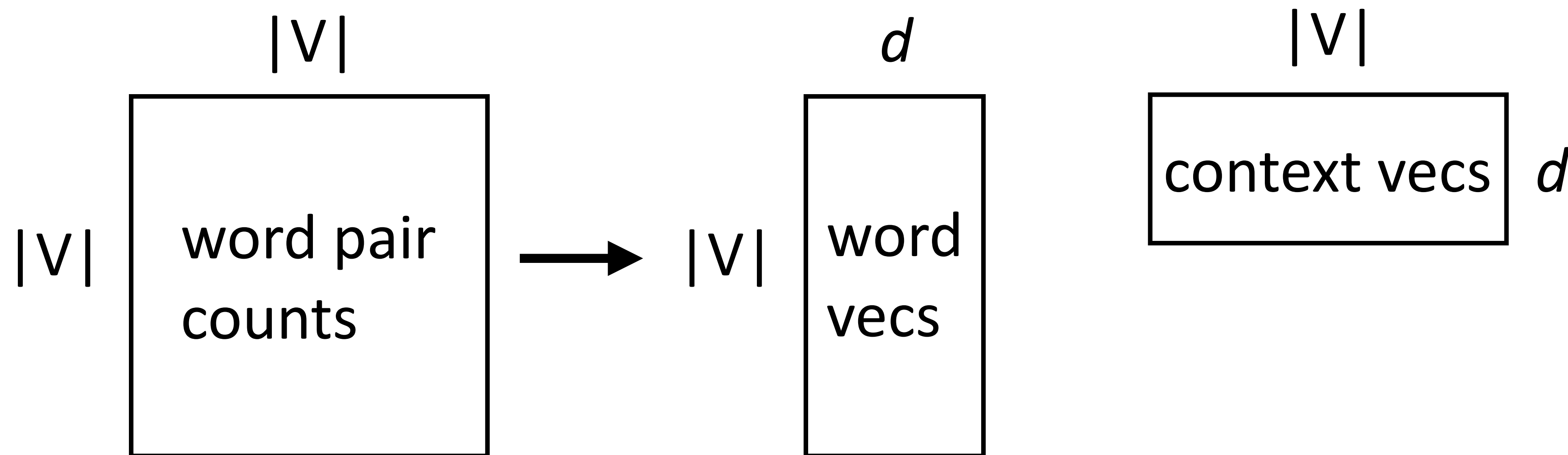
	knife	dog	sword	love	like
knife	0	1	6	5	5
dog	1	0	5	5	5
sword	6	5	0	5	5
love	5	5	5	0	5
like	5	5	5	5	2

Two words are “similar” in meaning if their context vectors are similar. Similarity == relatedness

# Connections with Matrix Factorization

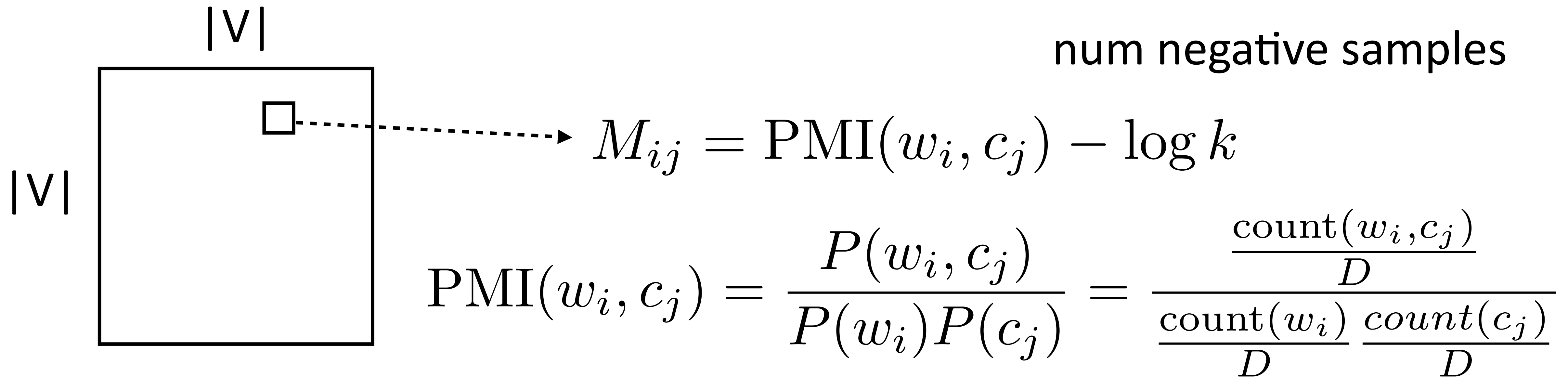
---

- ▶ Skip-gram model looks at word-word co-occurrences and produces two types of vectors



- ▶ Looks almost like a matrix factorization...can we interpret it this way?

# Skip-Gram as Matrix Factorization



Skip-gram objective *exactly* corresponds to factoring this matrix:

- ▶ If we sample negative examples from the uniform distribution over words
- ▶ ...and it's a *weighted* factorization problem (weighted by word freq)



# Co-occurrence Matrix

---

- ▶ Typical problems in word-word co-occurrences:
  - ▶ Raw frequency is not the best measure of association between words.
  - ▶ Frequent words are often more important than rare words that only appear once or twice;
  - ▶ But, frequent words (e.g., *the*) that appear in all documents are also not very useful signal.
- ▶ Solutions — weighing terms in word-word/word-doc co-occurrence matrix
  - ▶  $Tf*idf$
  - ▶ PPMI (Positive PMI)

# Co-occurrence Matrix

- ▶ Tf\*idf

- ▶ Tf: term frequency

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

word-doc co-occurrences

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	17
solider	2	80	62	89
fool	36	58	1	4
clown	20	15	2	3

- ▶ Idf: inverse document frequency

$$idf_i = \log_{10}\left(\frac{N}{df_i}\right)$$

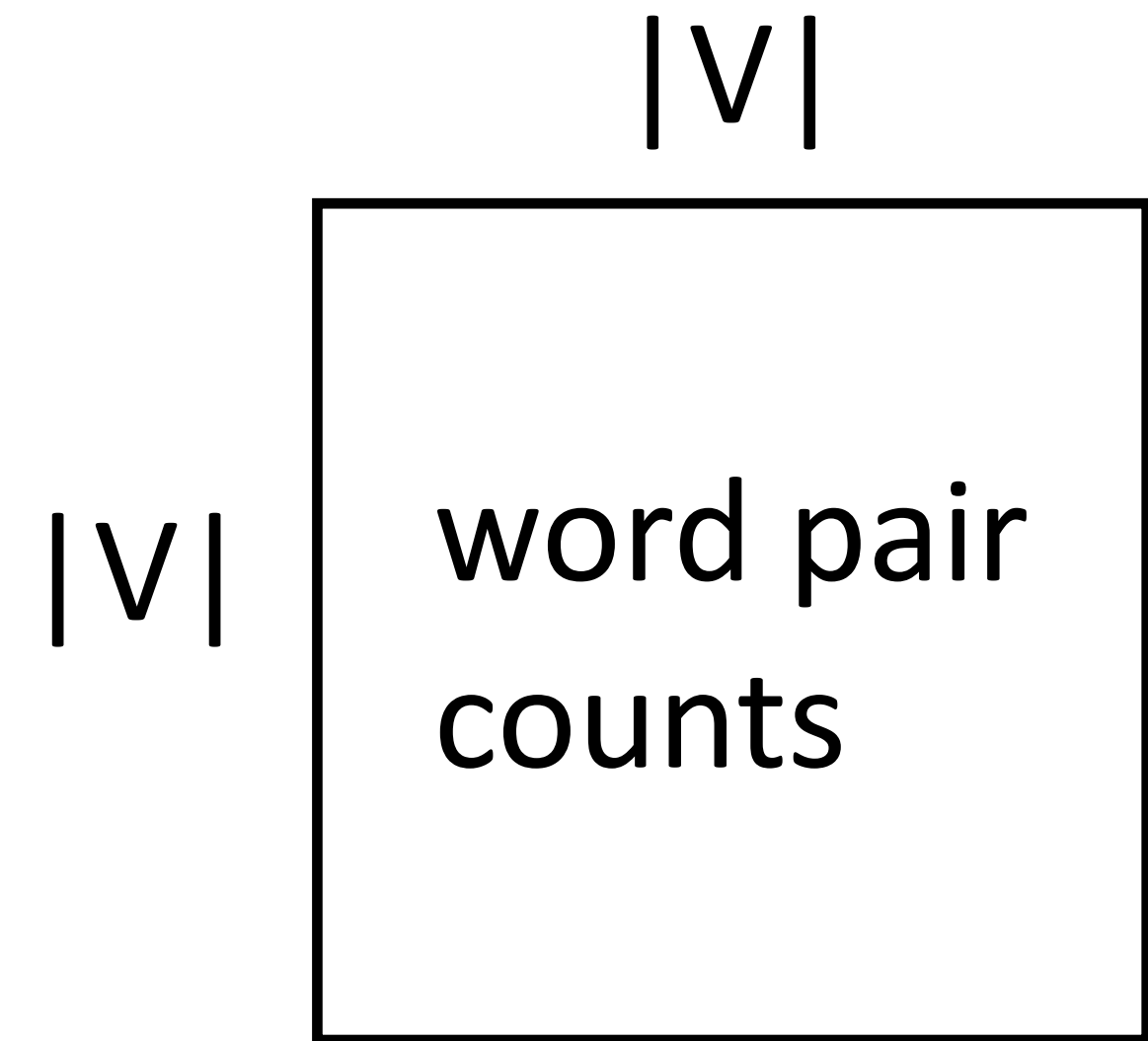
Total number of docs  
in collection

number of docs that  
have word i

# GloVe (Global Vectors)

---

- ▶ Also operates on counts matrix, weighted regression on the log co-occurrence matrix



- ▶ Loss 
$$= \sum_{i,j} f(\text{count}(w_i, c_j)) \left( w_i^\top c_j + a_i + b_j - \log \text{count}(w_i, c_j) \right)^2$$
- ▶ Constant in the dataset size (just need counts), quadratic in voc size
- ▶ By far the most common non-contextual word vectors used today (10000+ citations)

Pennington et al. (2014)

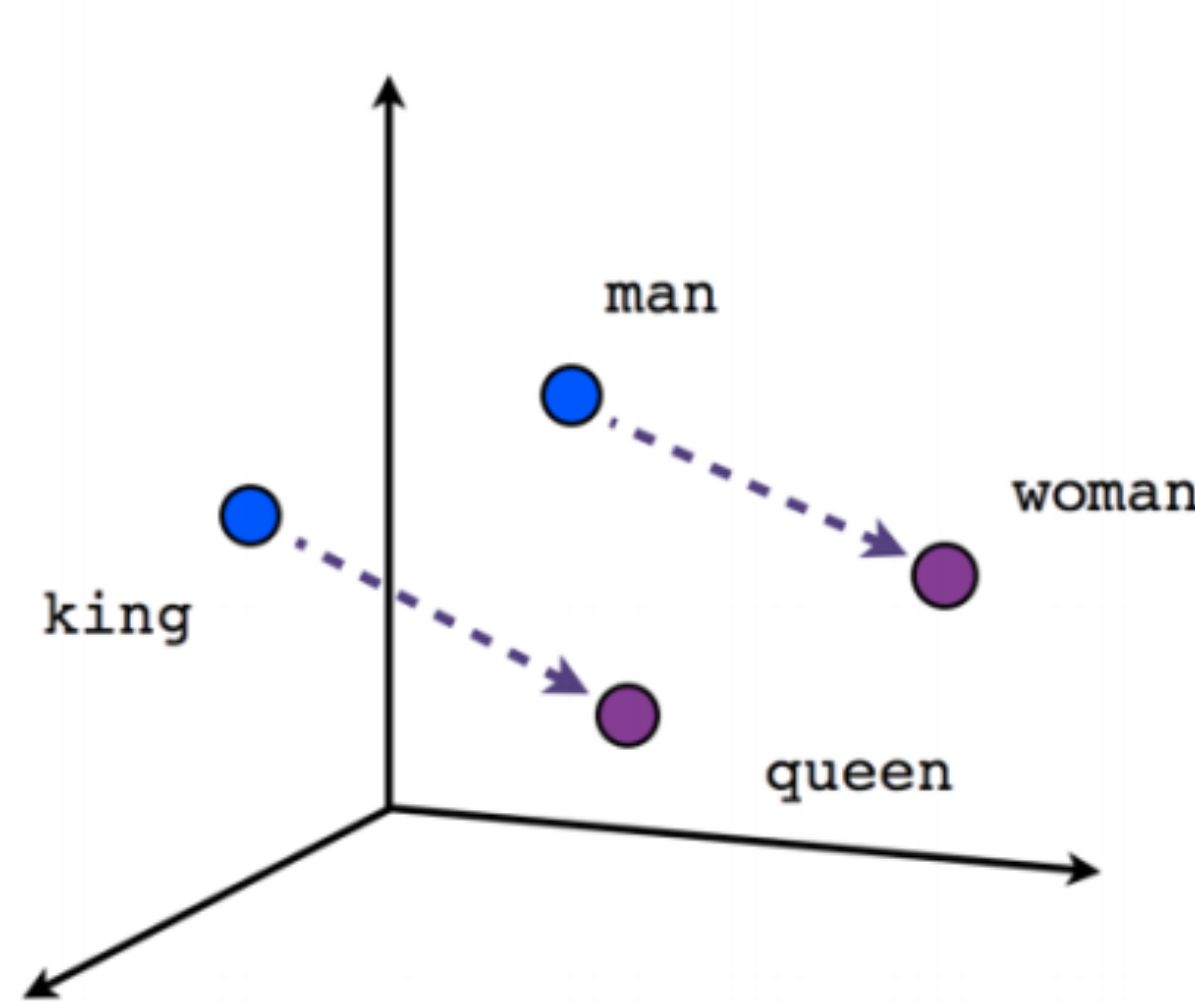
# Using Word Embeddings

---

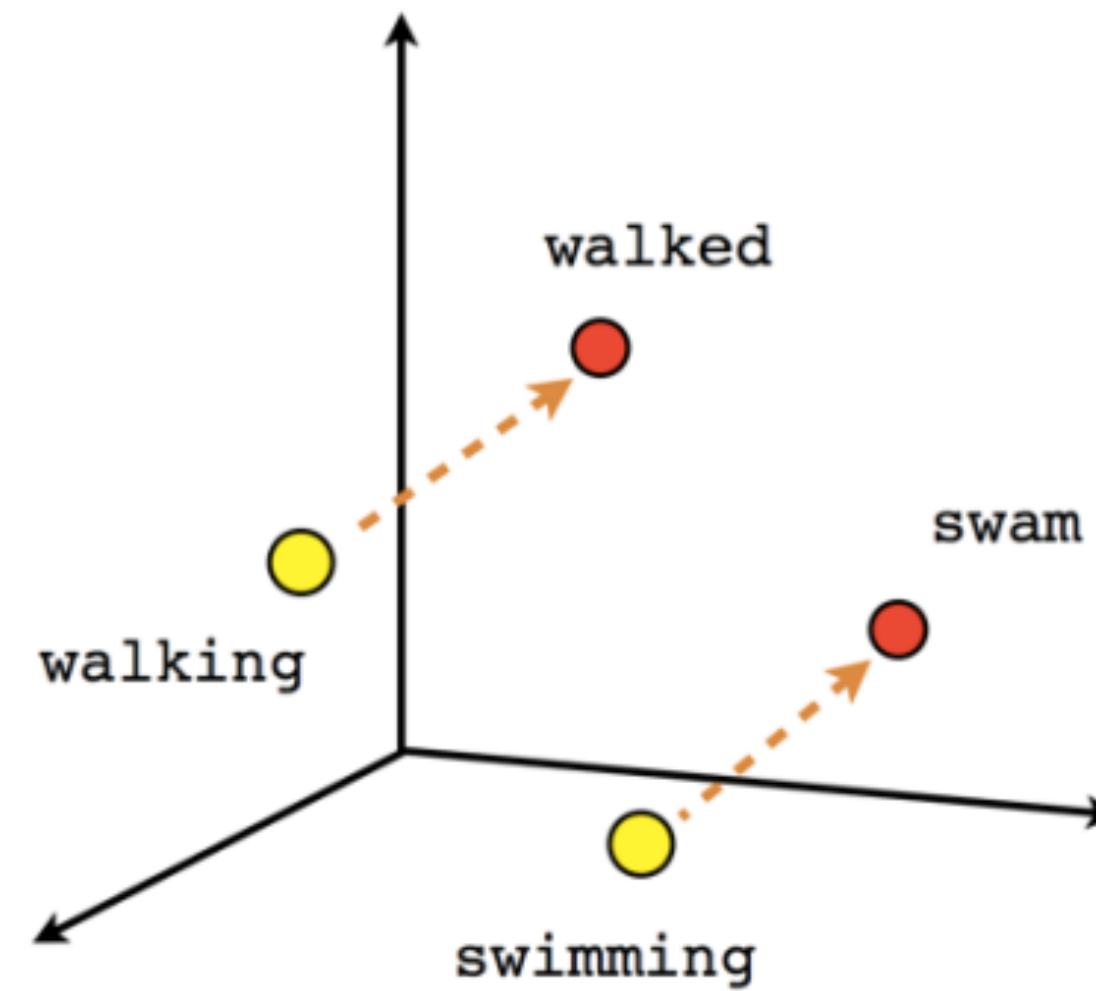
- ▶ Approach 1: learn embeddings as parameters from your data
  - ▶ Often works pretty well
- ▶ Approach 2: initialize using GloVe/word2vec/ELMo, keep fixed
  - ▶ Faster because no need to update these parameters
- ▶ Approach 3: initialize using GloVe, fine-tune
  - ▶ Works best for some tasks, not used for ELMo, often used for BERT

# Evaluation

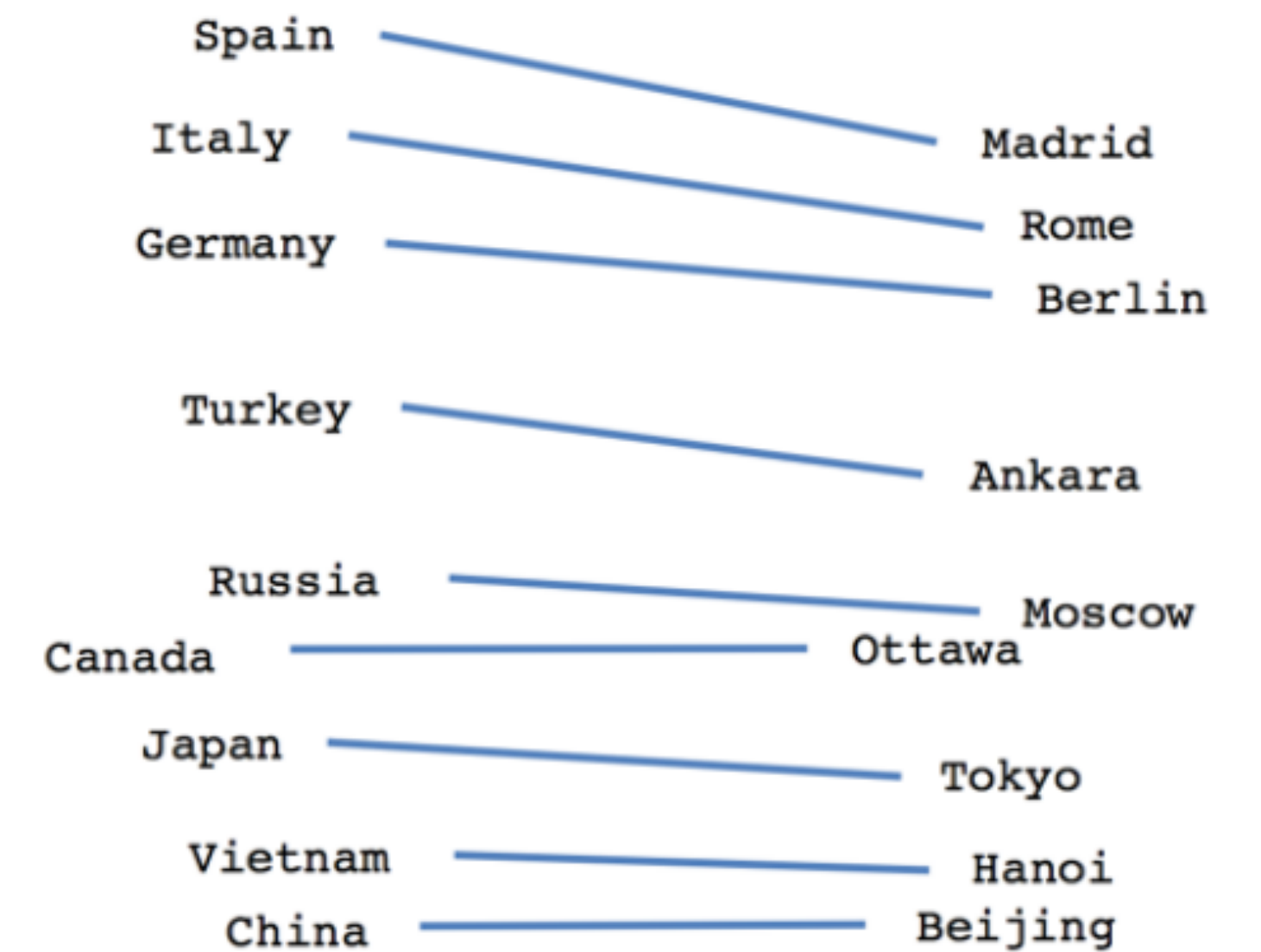
# Visualization



Male-Female



Verb tense



Country-Capital



# Visualization

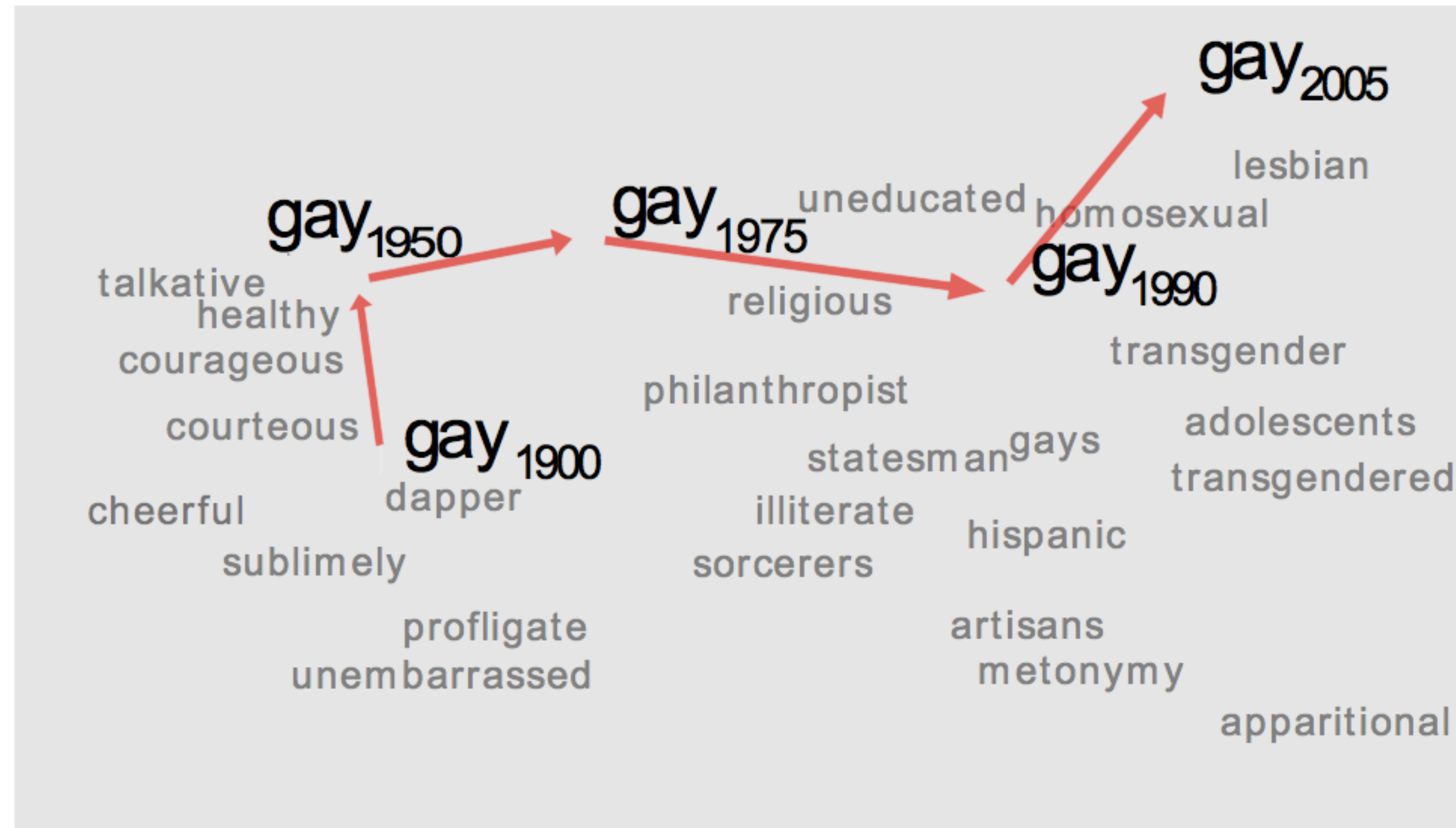


Figure 1: A 2-dimensional projection of the latent semantic space captured by our algorithm. Notice the semantic trajectory of the word **gay** transitioning meaning in the space.

# Evaluating Word Embeddings

---

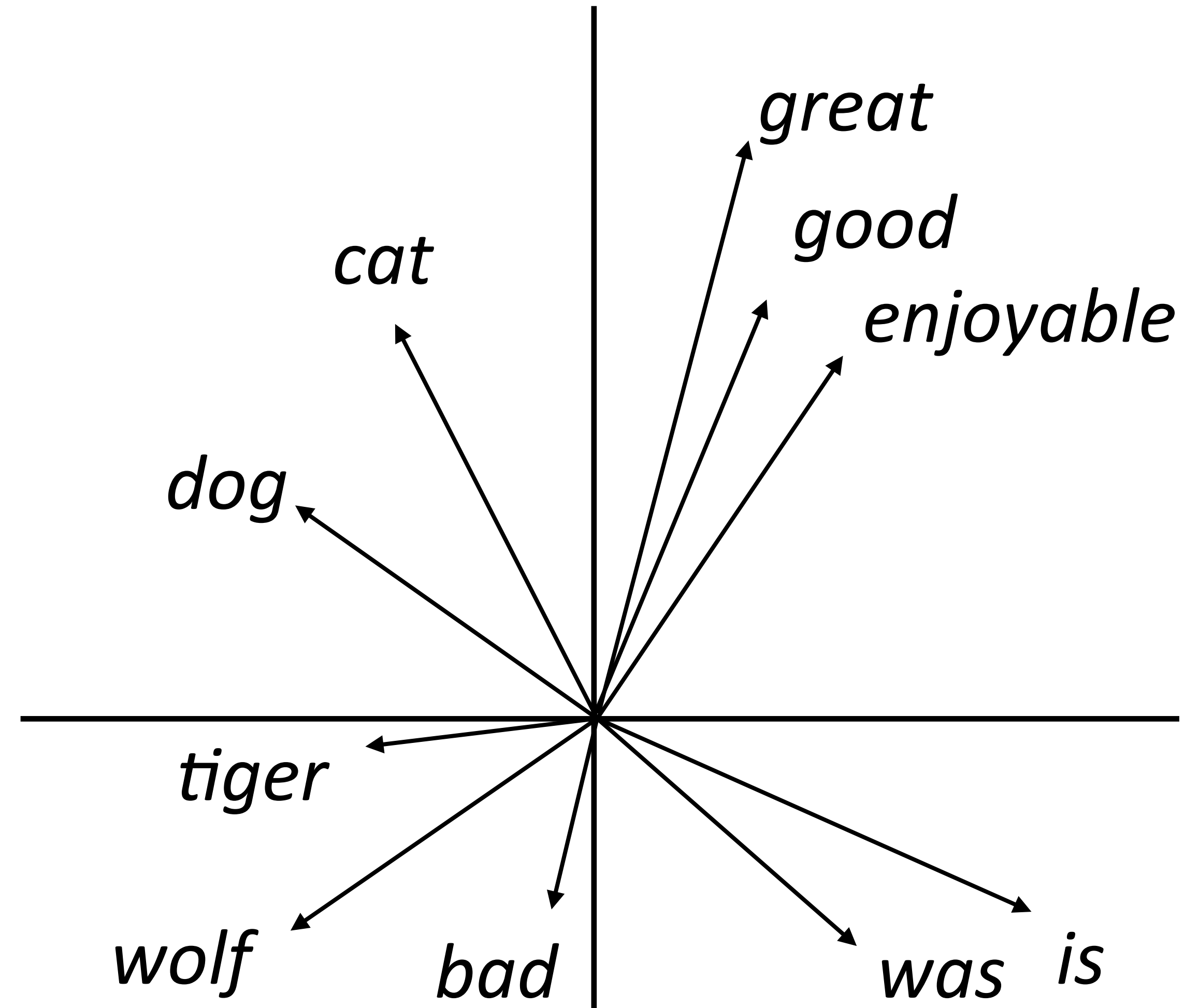
► What properties of language should word embeddings capture?

► Similarity: similar words are close to each other

► Analogy:

good is to best as smart is to ???

Paris is to France as Tokyo is to ???



# Word Similarity

---

## ► Cosine Similarity:

$$\text{cosine}(\vec{v}, \vec{w}) = \frac{\vec{v} \cdot \vec{w}}{|\vec{v}| |\vec{w}|} = \frac{\sum_{i=1}^N v_i w_i}{\sqrt{\sum_{i=1}^N v_i^2} \sqrt{\sum_{i=1}^N w_i^2}}$$

# Word Similarity

Word2vec →	Method	WordSim Similarity	WordSim Relatedness	Bruni et al. MEN	Radinsky et al. M. Turk	Luong et al. Rare Words	Hill et al. SimLex
	PPMI	.755	<b>.697</b>	.745	.686	.462	.393
	SVD	<b>.793</b>	.691	<b>.778</b>	.666	<b>.514</b>	.432
	SGNS	<b>.793</b>	.685	.774	<b>.693</b>	.470	<b>.438</b>
	GloVe	.725	.604	.729	.632	.403	.398

- ▶ SVD = singular value decomposition on PMI matrix
- ▶ GloVe does not appear to be the best when experiments are carefully controlled, but it depends on hyperparameters + these distinctions don't matter in practice



# Hypernymy Detection

- ▶ Hypernyms: detective *is a* person, dog *is a* animal
- ▶ Do word vectors encode these relationships?

Dataset	TM14	Kotlerman 2010	HypeNet	WordNet	Avg (10 datasets)
Random	52.0	30.8	24.5	55.2	23.2
Word2Vec + C	52.1	<b>39.5</b>	20.7	<b>63.0</b>	25.3
GE + C	53.9	36.0	21.6	58.2	26.1
GE + KL	52.0	39.4	23.7	54.4	25.9
DIVE + C· $\Delta$ S	<b>57.2</b>	36.6	<b>32.0</b>	60.9	<b>32.7</b>

- ▶ word2vec (SGNS) works barely better than random guessing here

Table 1: Comparison with other unsupervised embedding methods. The scores are AP@all (%) for the first 10 datasets and Spearman  $\rho$  (%) for HyperLex. Avg (10 datasets) shows the micro-average AP of all datasets except HyperLex. Word2Vec+C scores word pairs using cosine similarity on skip-grams. GE+C and GE+KL compute cosine similarity and negative KL divergence on Gaussian embedding, respectively.

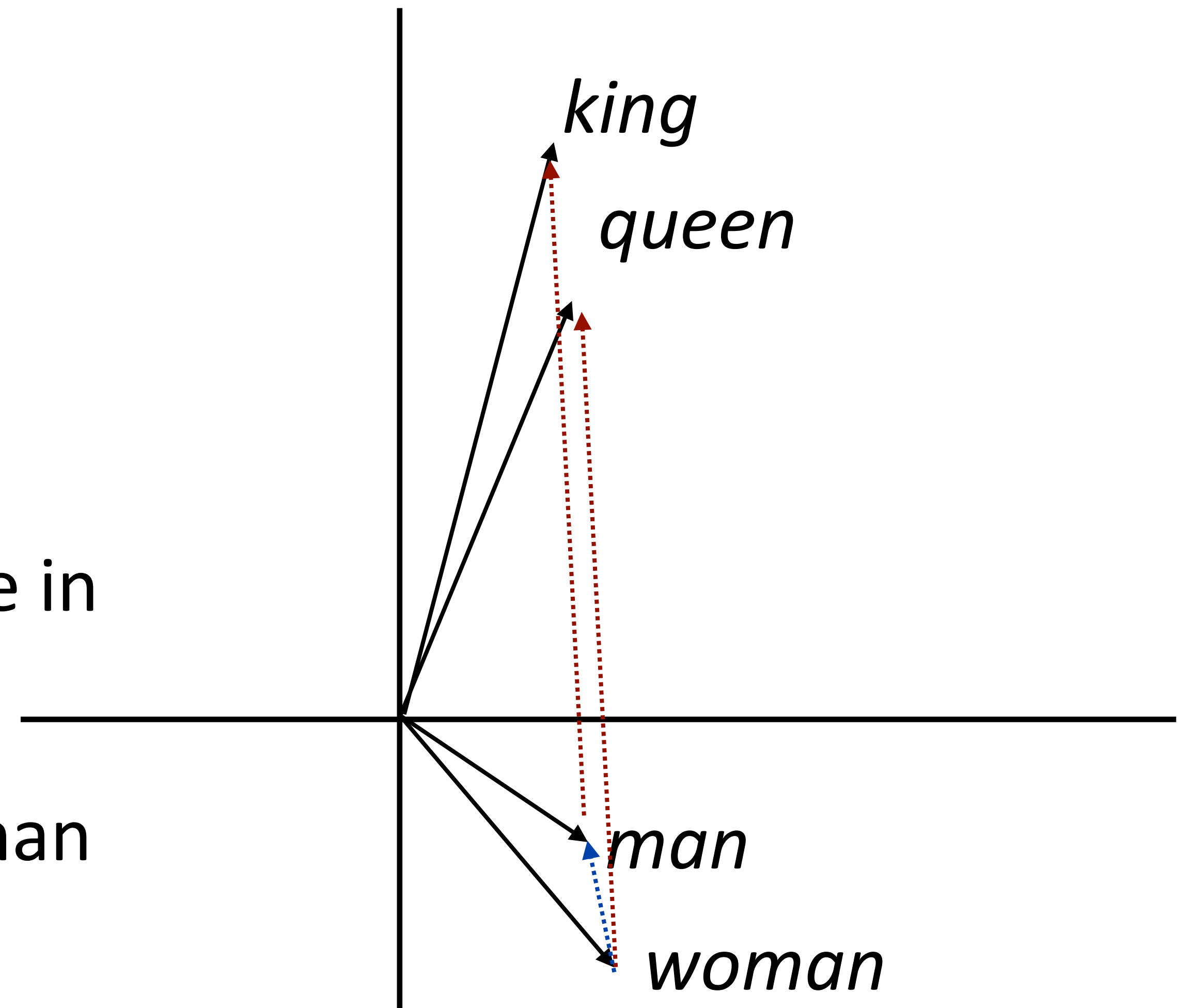
Chang et al. (2017)

# Analogies

$(king - man) + woman = queen$

$king + (woman - man) = queen$

- ▶ Why would this be?
- ▶ woman - man captures the difference in the contexts that these occur in
- ▶ Dominant change: more “he” with man and “she” with woman — similar to difference between king and queen





# Analogies

---

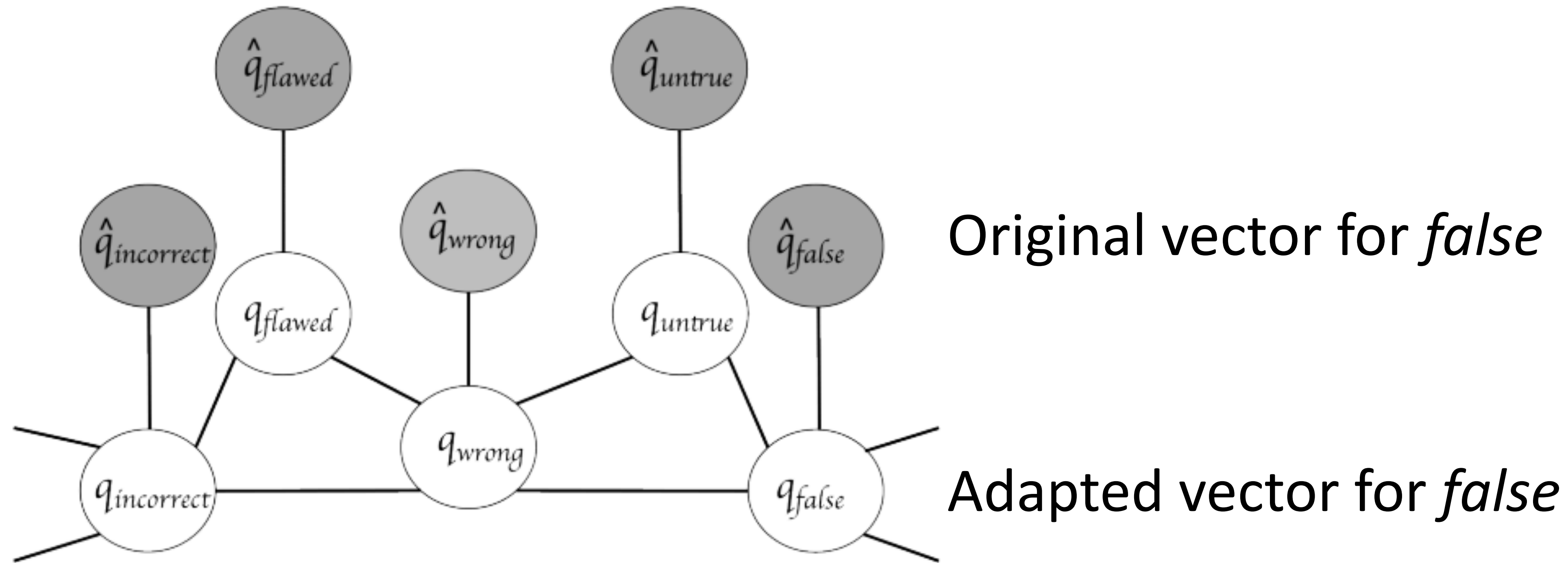
Method	Google	MSR
	Add / Mul	Add / Mul
PPMI	.553 / .679	.306 / .535
SVD	.554 / .591	.408 / .468
SGNS	.676 / <b>.688</b>	.618 / <b>.645</b>
GloVe	.569 / .596	.533 / .580

- ▶ These methods can perform well on analogies on two different datasets using two different methods

$$\text{Maximizing for } b: \text{Add} = \cos(b, a_2 - a_1 + b_1) \quad \text{Mul} = \frac{\cos(b_2, a_2) \cos(b_2, b_1)}{\cos(b_2, a_1) + \epsilon}$$

Levy et al. (2015)

# Using Semantic Knowledge



- ▶ Structure derived from a resource like WordNet
- ▶ Doesn't help most problems

# Takeaways

---

- ▶ Word vectors: learning word  $\rightarrow$  context mappings has given way to matrix factorization approaches (constant in dataset size)
- ▶ Lots of pretrained embeddings work well in practice, they capture some desirable properties
- ▶ Even better: context-sensitive word embeddings (ELMo/BERT/etc.) — will talk later in the semester
- ▶ Next time: sequence modeling, HMM, ...