Word Embeddings

Wei Xu

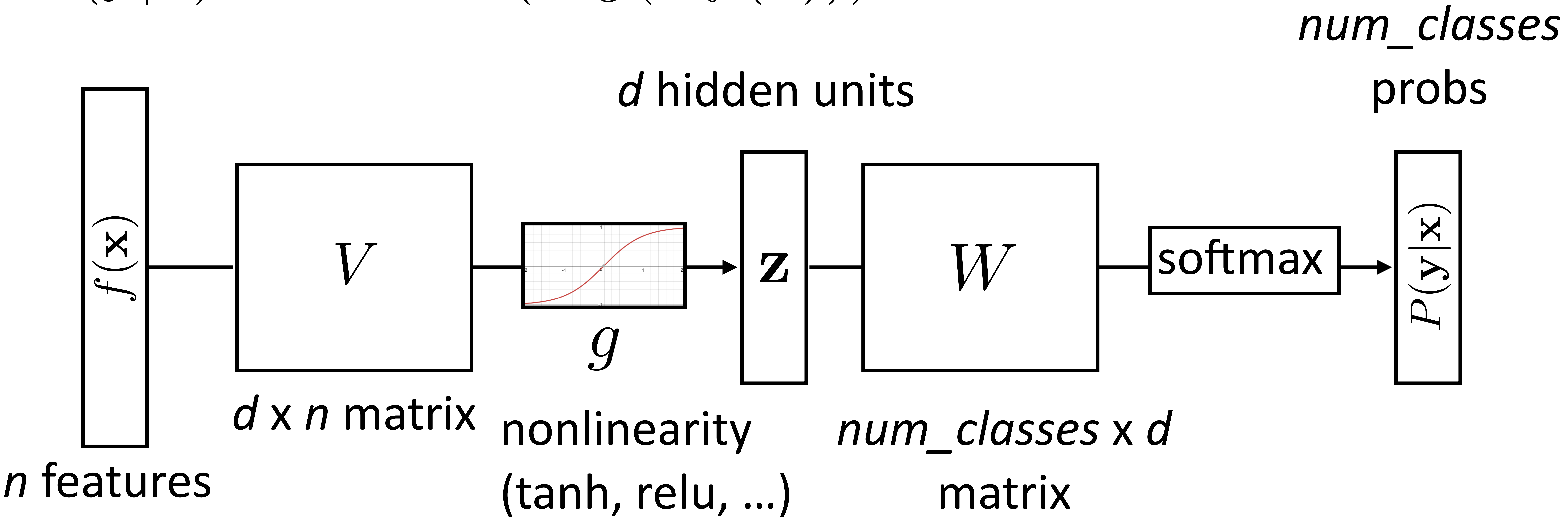
(many slides from Greg Durrett)

This Lecture

- Word representations
- word2vec/GloVe
- Reading: Eisenstein 3.3.4, 14.5, 14.6, J+M 6, Goldberg 5

Recap: Neural Networks for Classification

$$P(\mathbf{y}|\mathbf{x}) = \mathrm{softmax}(Wg(Vf(\mathbf{x})))$$

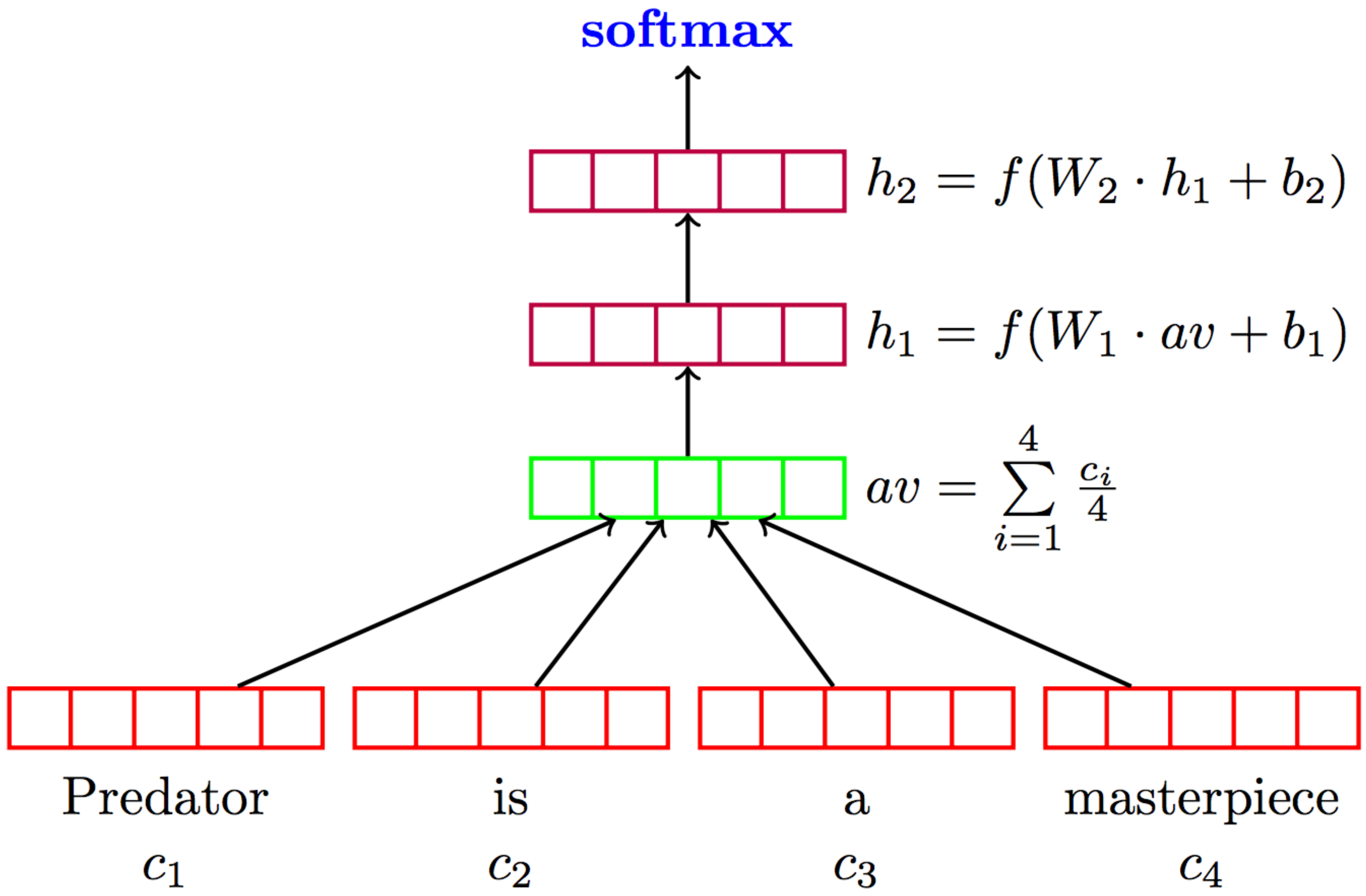


We can think of a neural network classifier with one hidden layer as building a vector z which is a hidden layer representation (i.e. latent features) of the input, and then running standard logistic regression on the features that the network develops in z.

Word Representations

Sentiment Analysis

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



Iyyer et al. (2015)

Word Embeddings

- Want a vector space where similar words have similar embeddings

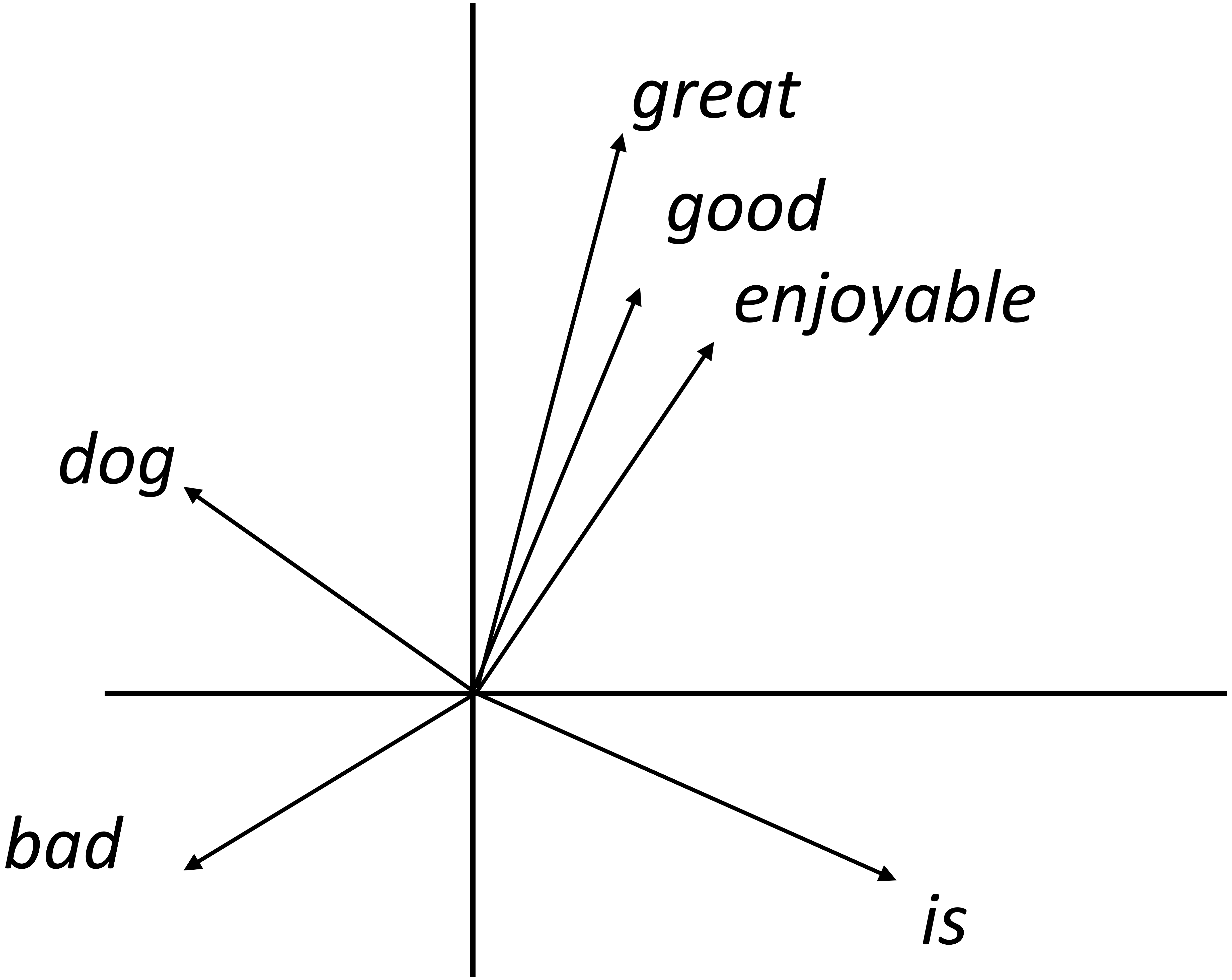
the movie was great

≈

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

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

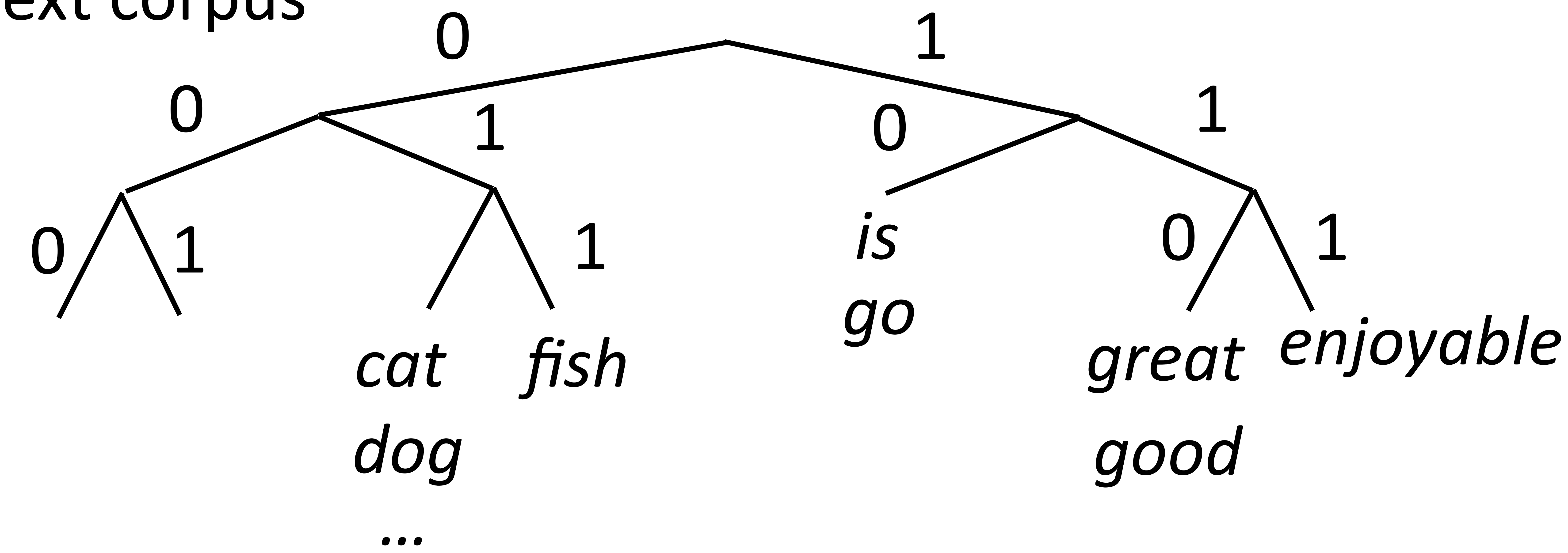
A bottle of ***tesgüino*** is on the table
Everybody likes ***tesgüino***
Tesgüino makes you drunk
We make ***tesgüino*** out of corn.


John R. Firth

slide credit: Dan Klein, Dan Jurafsky

Discrete Word Representations

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



- 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 Neural Networks

Brown et al. (1992)

Discrete Word Representations

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

mailman	100000110100111
salesman	1000001101110000
bookkeeper	10000011011100010
troubleshooter	100000110111000110
bouncer	100000110111000111
technician	10000011011100100
janitor	10000011011100101
saleswoman	10000011011100110
...	
Nike	1011011100100101011100
Maytag	10110111001001010111010
Generali	10110111001001010111011
Gap	1011011100100101011110
Harley-Davidson	10110111001001010111110
Enfield	101101110010010101111110
genus	101101110010010101111111
Microsoft	101101110010010110000
Ventritex	1011011100100101110010
Tractebel	10110111001001011100110
Synopsys	10110111001001011100111
WordPerfect	10110111001001011101000
....	
John	101110010000000000
Consuelo	101110010000000001
Jeffrey	101110010000000010
Kenneth	10111001000000001100
Phillip	101110010000000011010
WILLIAM	101110010000000011011
Timothy	10111001000000001110

word cluster features (bit string prefix)

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 $c_1, c_2, c_3, \ldots c_k$
- For $i = (k+1) \ldots |V|$
  - Create a new cluster c_{k+1} *(we have $k+1$ clusters)*
  - Choose two clusters from $k+1$ clusters based on quality(C) and merge *(back to k clusters)*

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

mutual information between adjacent clusters

entropy of the word distribution

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

Learn more: Percy Liang's phd thesis - Semi-Supervised Learning for Natural Language

Word Representations

- Count-based: tf*idf, PPMI, …
- Class-based: Brown Clusters, …
- Distributed prediction-based embeddings: Word2vec (2013), GloVe (2014), FastText, …
- Distributed contextual embeddings: ELMo (2018), BERT (2019), GPT, …
- + many more variants: multi-sense embeddings, syntactic embeddings, …

Neural Probabilistic Language Model

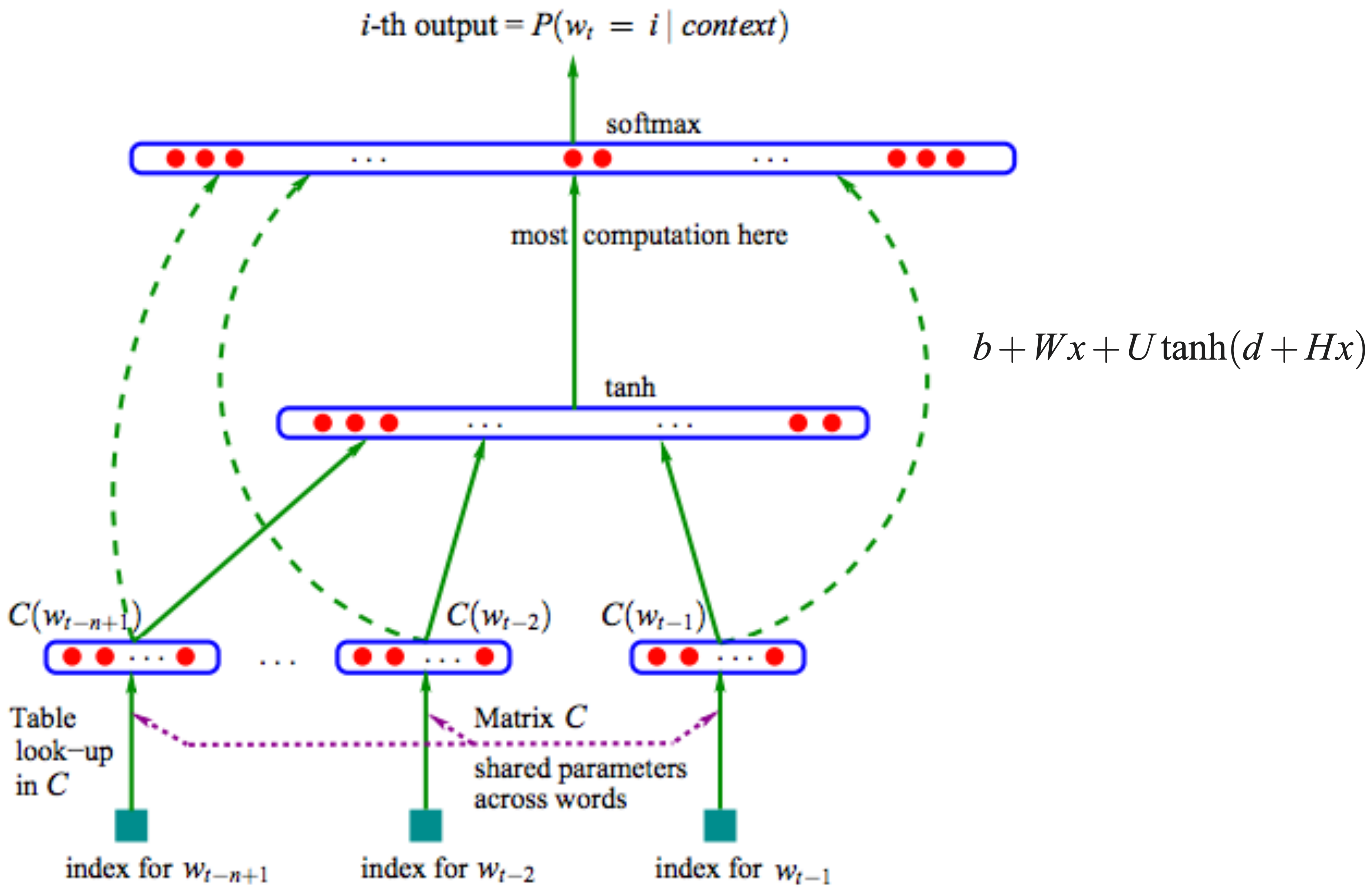


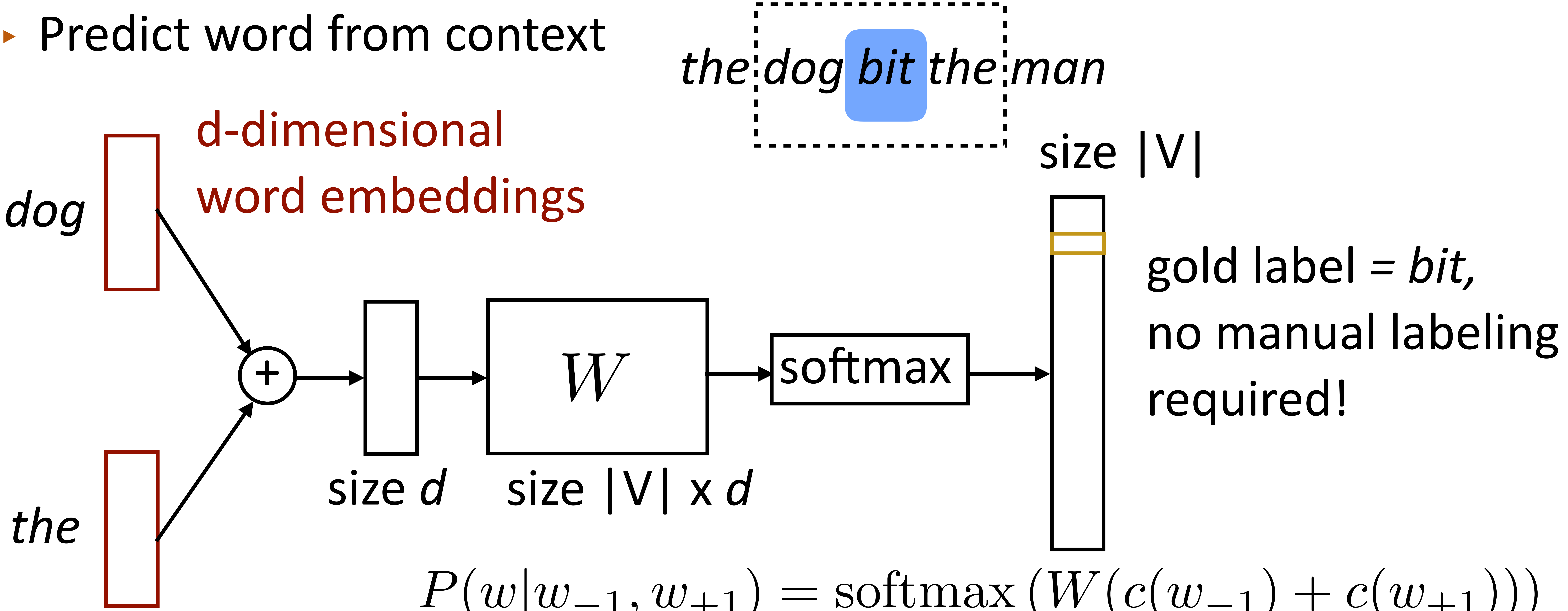
Figure 1: Neural architecture: $f(i, w_{t-1}, \cdots, w_{t-n+1}) = g(i, C(w_{t-1}), \cdots, C(w_{t-n+1}))$ where g is the neural network and $C(i)$ is the i-th word feature vector.

Bengio et al. (2003)

word2vec/GloVe

word2vec: Continuous Bag-of-Words

- Predict word from context



$$P(w|w_{-1}, w_{+1}) = \text{softmax}\left(W(c(w_{-1}) + c(w_{+1}))\right)$$

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

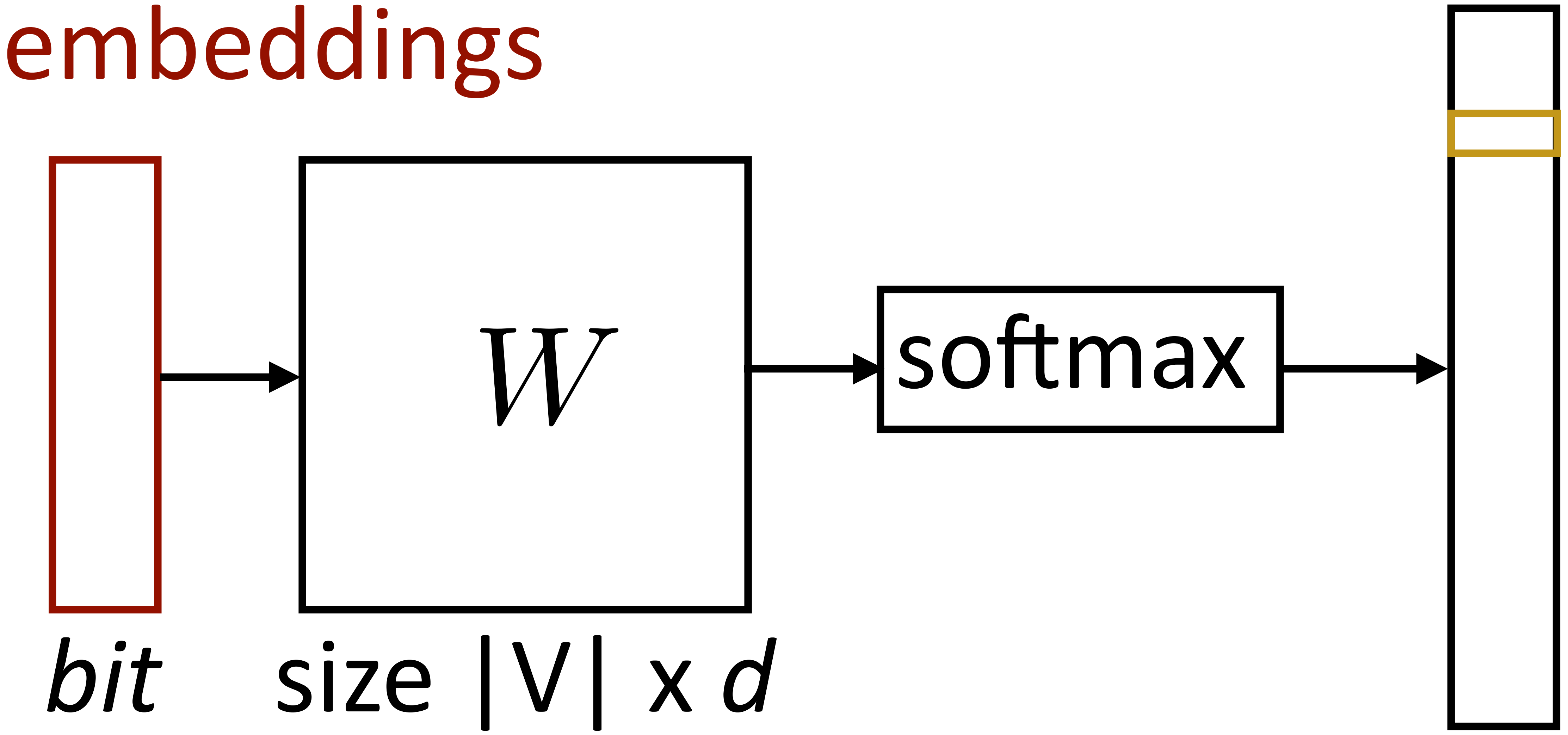
Mikolov et al. (2013)

word2vec: Skip-Gram

- Predict one word of context from word

the dog bit the man

d-dimensional
word embeddings



gold label = *dog*

$$P(w'|w) = \text{softmax}(We(w))$$

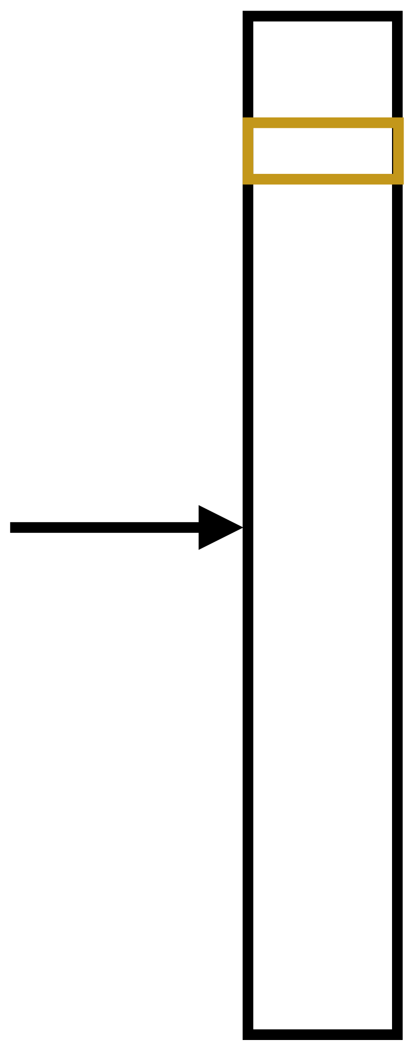
- Another training example: *bit* -> *the*
- Parameters: *d* x |V| vectors, |V| x *d* output parameters (W) (also usable as vectors!)

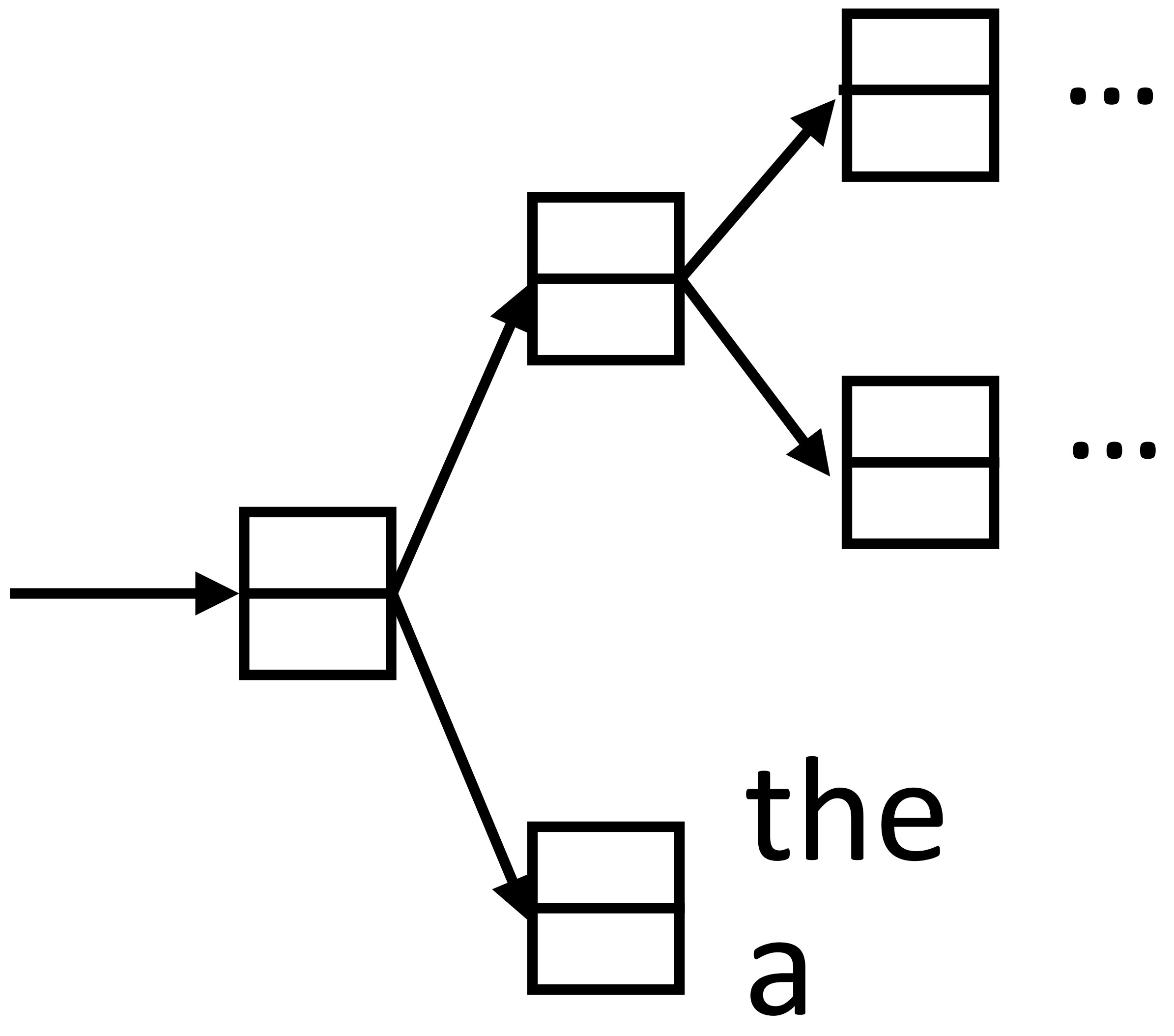
Mikolov et al. (2013)

Hierarchical Softmax

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

- 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*,

Mikolov et al. (2013)

http://building-babylon.net/2017/08/01/hierarchical-softmax/

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, dog*) => +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* x |V| vectors, *d* x |V| context vectors (same # of params as before)

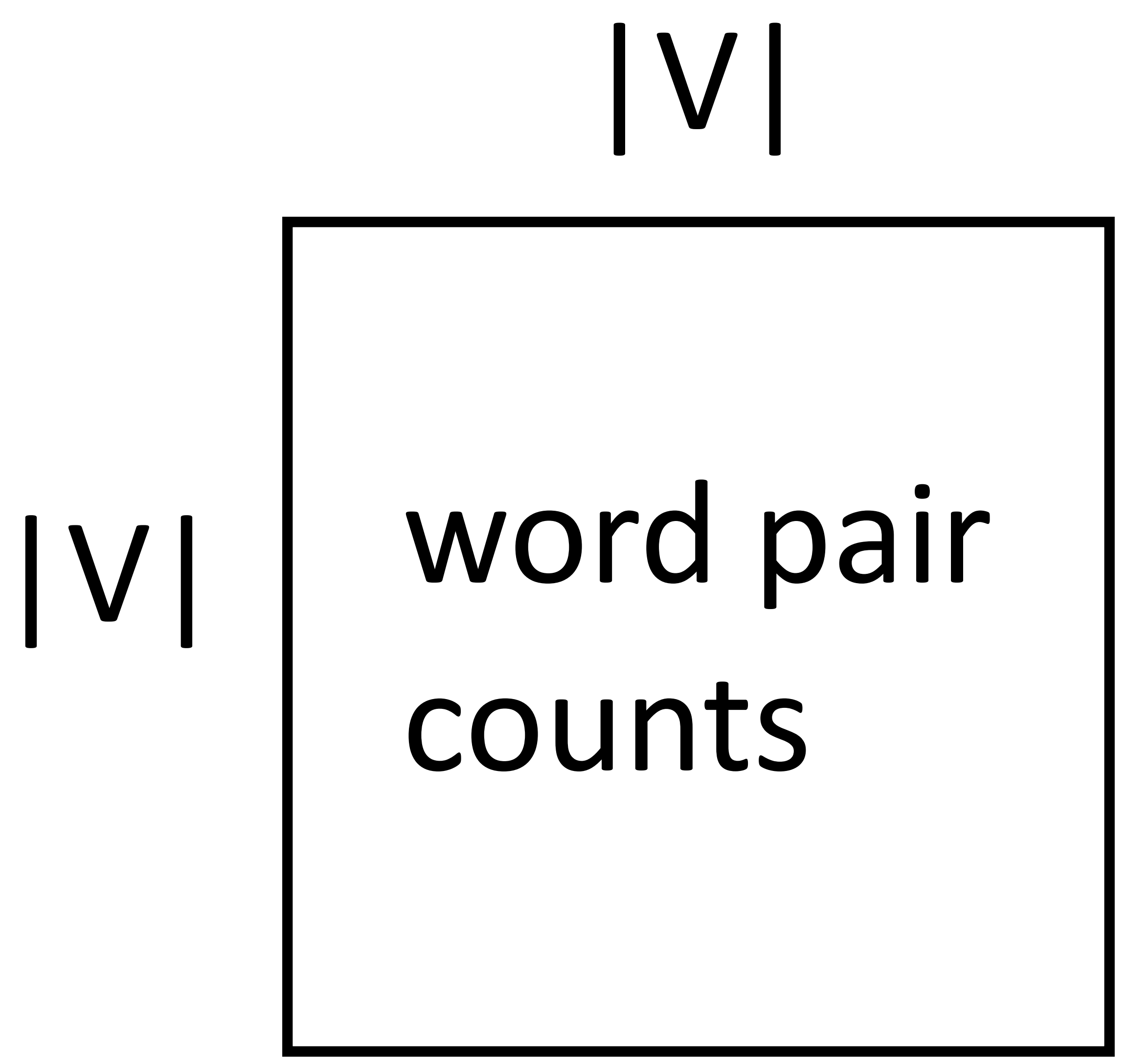
- Objective = $\log P(y = 1|w, c) + \sum_{i=1}^{k} \log P(y = 0|w_i, c)$

sampled

Mikolov et al. (2013)

Connections with Matrix Factorization

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



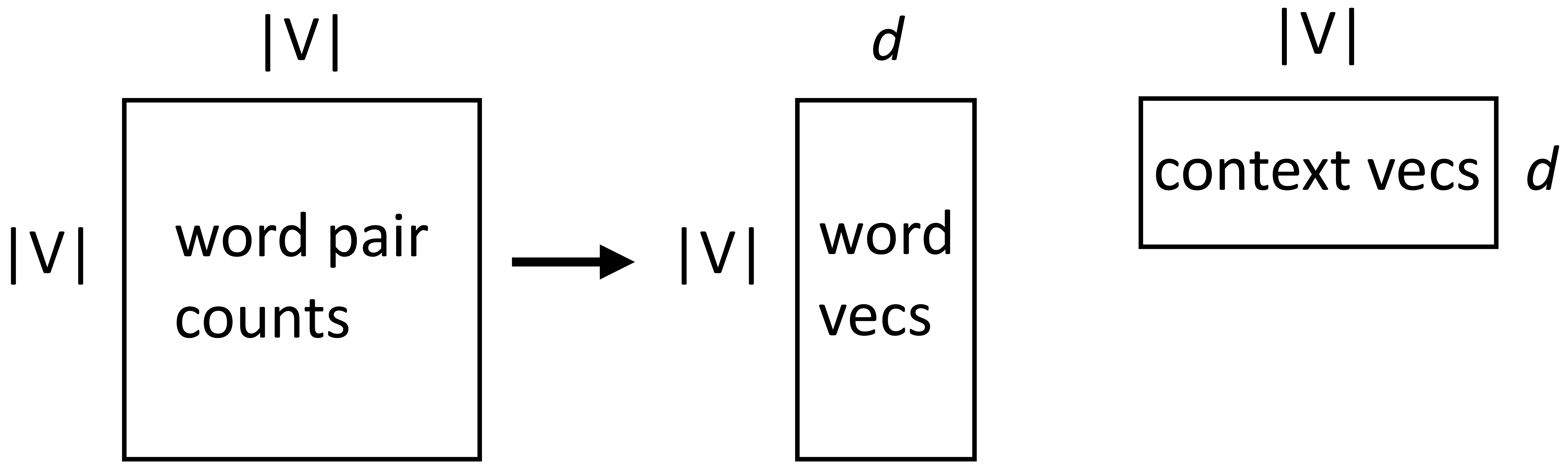
	knife	dog	sword	love	like
knife	0	1	6	5	5
dog	1	0	5	5	5
sword	6	5	0	5	5
love	5	5	5	0	5
like	5	5	5	5	2

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

Levy et al. (2014)

Connections with Matrix Factorization

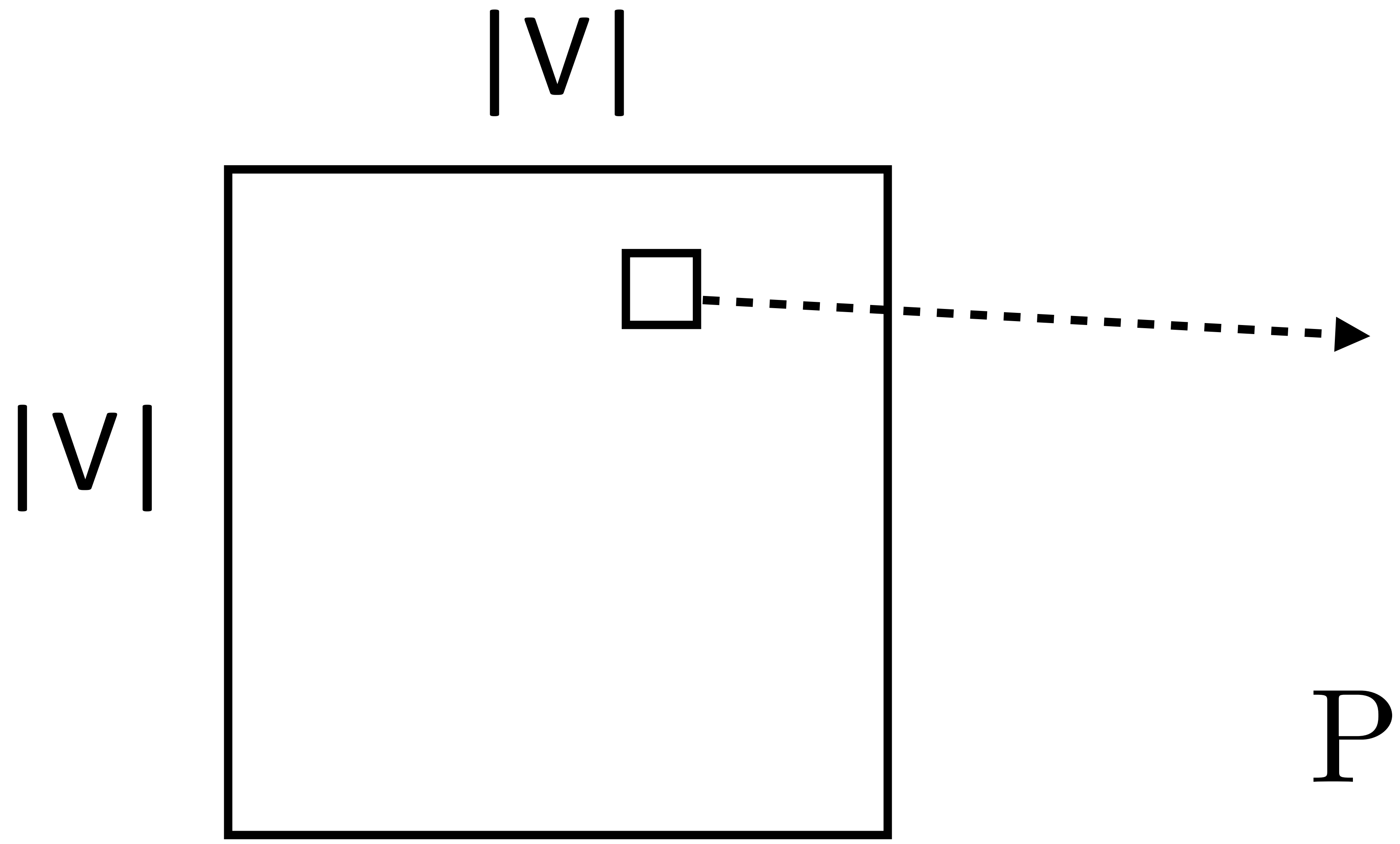
- Skip-gram model looks at word-word co-occurrences and produces two types of vectors



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

Levy et al. (2014)

Skip-Gram as Matrix Factorization



num negative samples

$$M_{ij} = \mathrm{PMI}(w_i, c_j) - \log k$$

$$\mathrm{PMI}(w_i, c_j) = \frac{P(w_i, c_j)}{P(w_i)P(c_j)}$$

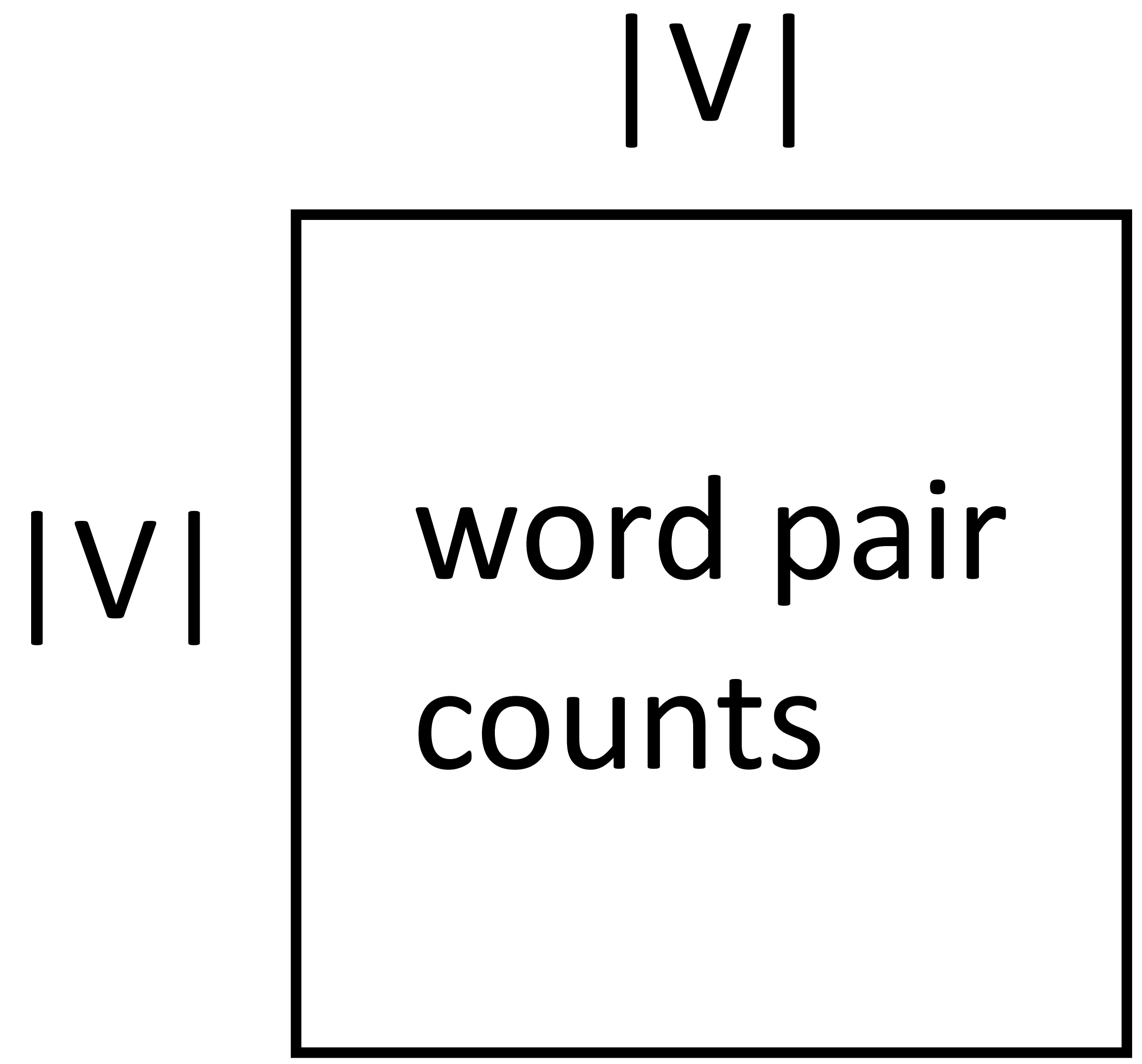
Skip-gram objective *exactly* corresponds to factoring this matrix:

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

Levy et al. (2014)

GloVe (**Glo**bal **Ve**ctors)

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



- Objective = $\sum_{i,j} f(\mathrm{count}(w_i, c_j)) \left(w_i^\top c_j + a_i + b_j - \log \mathrm{count}(w_i, c_j)\right)^2$
- Constant in the dataset size (just need counts), quadratic in voc size
- By far the most common non-contextual word vectors used today (30000+ citations)

Pennington et al. (2014)

Using Word Embeddings

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

NER in Twitter

Brown clusters

2m 2ma 2mar 2mara 2maro 2marrow 2mor 2mora 2moro 2morow 2morr 2morro 2morrow 2moz 2mr 2mro 2mrrw 2mrw 2mw tmmrw tmo tmoro tmorrow tmoz tmr tmro tmrow tmrrow tmrrw tmrw tmrww tmw tomaro tomarow tomarro tomarrow tomm tommarow tommarrow tommoro tommorow tommorrow tommorw tommrow tomo tomolo tomoro tomorow tomorro tomorrw tomoz tomrw tomz

Ritter et al. (2011)

Word2vec

Both

System	Fin10Dev	Rit11	Fro14	Avg
CoNLL	27.3	27.1	29.5	28.0
+ Brown	38.4	39.4	42.5	40.1
+ Vector	40.8	40.4	42.9	41.4
+ Reps	42.4	42.2	46.2	43.6
Fin10	36.7	29.0	30.4	32.0
+ Brown	59.9	53.9	56.3	56.7
+ Vector	61.5	56.4	58.4	58.8
+ Reps	64.0	58.5	60.2	60.9
CoNLL+Fin10	44.7	39.9	44.2	42.9
+ Brown	54.9	52.9	58.5	55.4
+ Vector	58.9	55.2	59.9	58.0
+ Reps	58.9	56.4	61.8	59.0
+ Weights	64.4	59.6	63.3	62.4

Table 5: Impact of our components on Twitter NER performance, as measured by F1, under 3 data scenarios.

Cherry & Guo (2015)

Visualization

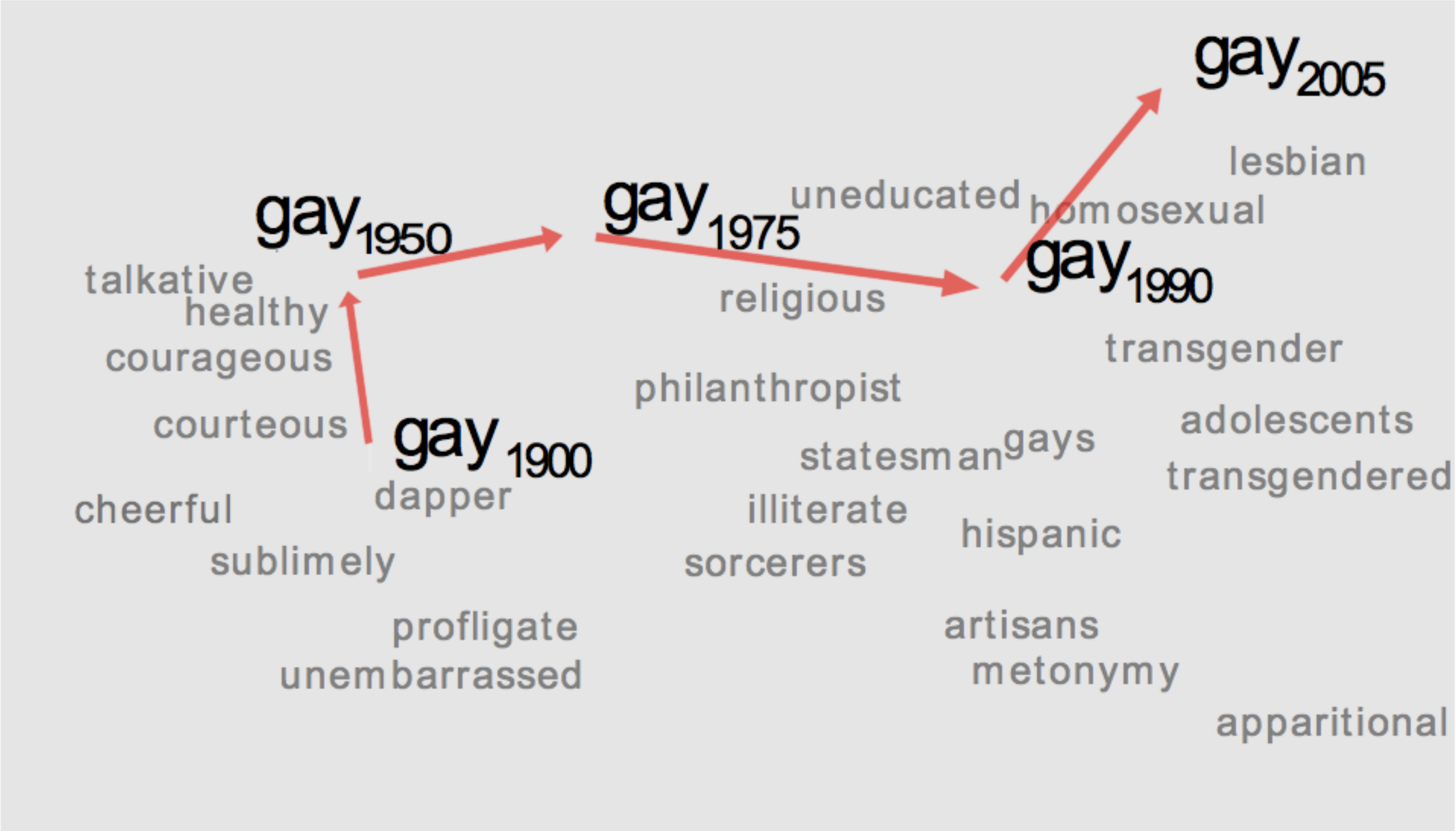


Figure 1: A 2-dimensional projection of the latent semantic space captured by our algorithm. Notice the semantic trajectory of the word **gay** transitioning meaning in the space.

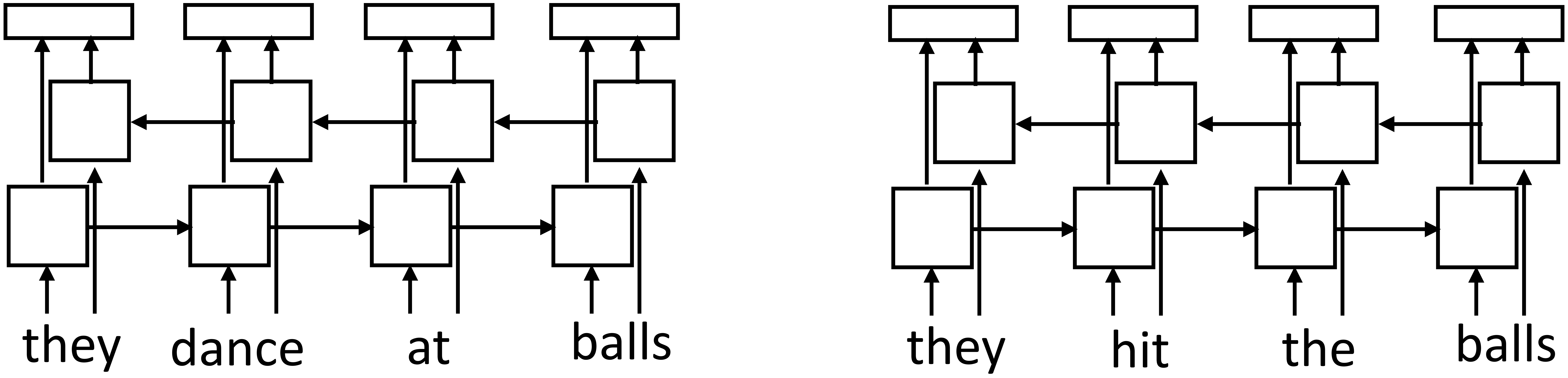
Kulkarni et al. (2015)

Takeaways

- Word vectors: learning word -> 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, …

Preview: Context-dependent Embeddings

- How to handle different word senses? One vector for *balls*



- Train a neural language model to predict the next word given previous words in the sentence, use its internal representations as word vectors
- *Context-sensitive* word embeddings: depend on rest of the sentence
- *Huge* improvements across nearly all NLP tasks over word2vec & GloVe

Peters et al. (2018)