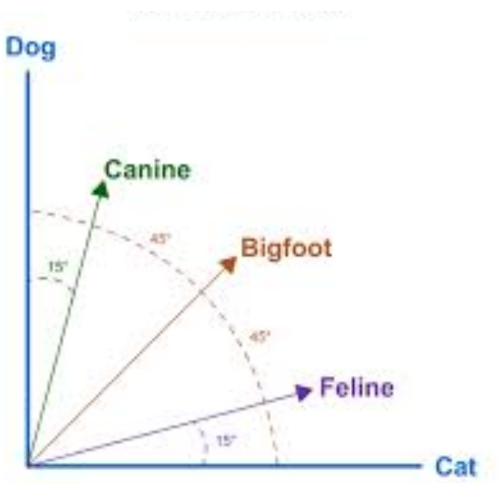
Distributional Semantics

COMP90042

Natural Language Processing

Lecture 10



Lexical Databases - Problems

- Manually constructed
  - Expensive
  - Human annotation can be biased and noisy
- Language is dynamic
  - New words: slang, terminology, etc.
  - New senses
- The Internet provides us with massive amounts of text. Can we use that to obtain word meanings?

Distributional Hypothesis

- "You shall know a word by the company it keeps" (Firth, 1957)
- Document co-occurrence often indicative of topic (*document* as context)
  - E.g. *voting* and *politics*
- Local context reflects a word's semantic class (*word window* as context)
  - E.g. *eat a pizza*, *eat a burger*

Guessing Meaning from Context

- Learn unknown word from its usage
- *E.g., tezgüino*

(14.1) A bottle of _____ is on the table.

(14.2) Everybody likes _____.

(14.3) Don't have _____ before you drive.

(14.4) We make _____ out of corn.

- Look at other words in same (or similar) contexts

	(14.1)	(14.2)	(14.3)	(14.4)	...
tezgüino	1	1	1	1	
loud	0	0	0	0	
motor oil	1	0	0	1	
tortillas	0	1	0	1	
choices	0	1	0	0	
wine	1	1	1	0	

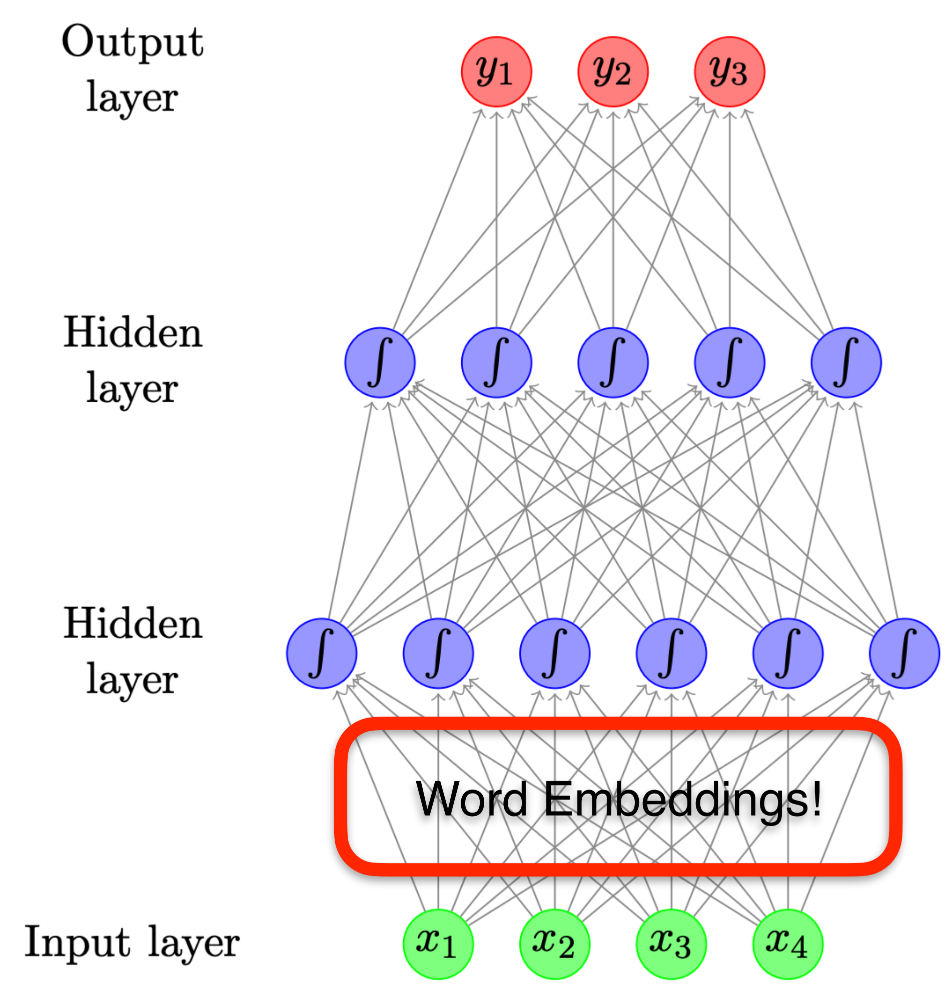
Word Vectors

	(14.1)	(14.2)	(14.3)	(14.4)	...
tezgüino	1	1	1	1	
loud	0	0	0	0	
motor oil	1	0	0	1	
tortillas	0	1	0	1	
choices	0	1	0	0	
wine	1	1	1	0	

- Each row can be thought of a **word vector**
- It describes the distributional properties
- Capture all sorts of semantic relationships (synonymy, analogy, etc)

Word Embeddings?

- We've seen word vectors before: word embeddings!
- Here we will learn other ways to produce word vectors
  - Count-based methods
  - More efficient neural methods designed just for learning word vectors



Count-Based Methods

The Vector Space Model

- Fundamental idea: represent meaning as a vector
- Consider documents as context
- One matrix, two viewpoints
  - Documents represented by their words
  - Words represented by their documents

	…	state	fun	heaven	…
…					
425		0	1	0	
426		3	0	0	
427		0	0	0	
……					

Manipulating the VSM

- Weighting the values (beyond frequency)
- Creating low-dimensional dense vectors
- Comparing vectors

Tf-idf

- Standard weighting scheme for information retrieval

$$idf_w = \log \frac{|D|}{df_w}$$

- Discounts common words

	…	the	country	hell	…
…					
425		43	5	1	
426		24	1	0	
427		37	0	3	
…					
df		500	14	7	

tf matrix

	…	the	country	hell	…
…					
425		0	26.0	6.2	
426		0	5.2	0	
427		0	0	18.6	
…					

tf-idf matrix

Dimensionality Reduction

- Term-document matrices are very **sparse**
- Dimensionality reduction: create shorter, denser vectors
- More practical (less features)
- Remove noise (less overfitting)

Singular Value Decomposition

$$
\begin{array}{cc}
 & |D| \\
|V| & \begin{bmatrix} 0 & 1 & 0 & & 0 \\ 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & & 0 \\ & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix}
\end{array}
$$

A
(term-document matrix)

$$A = U \Sigma V^T$$

U
(new term matrix)

Σ
(singular values)

V^T
(new document matrix)

$$
\begin{array}{cc}
 & m \\
|V| & \begin{bmatrix} 2.2 & 0.3 & & 8.7 \\ 5.5 & -2.8 & \cdots & 0.1 \\ -1.3 & 3.7 & & 3.5 \\ \vdots & & \ddots & \vdots \\ 2.9 & -2.1 & \cdots & -1.9 \end{bmatrix}
\end{array}
\quad
\begin{array}{cc}
 & m \\
m & \begin{bmatrix} 9.1 & 0 & 0 & & 0 \\ 0 & 4.4 & 0 & \cdots & 0 \\ 0 & 0 & 2.3 & & 0 \\ & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 0.1 \end{bmatrix}
\end{array}
\quad
\begin{array}{cc}
 & |D| \\
m & \begin{bmatrix} -0.2 & 4.0 & & -1.3 \\ -4.1 & 0.6 & \cdots & -0.2 \\ 2.6 & 6.1 & & 1.4 \\ \vdots & & \ddots & \vdots \\ -1.9 & -1.8 & \cdots & 0.3 \end{bmatrix}
\end{array}
$$

$m = Rank(A)$

Truncating – Latent Semantic Analysis

- Truncating *U, Σ,* and *V* to *k* dimensions produces best possible *k* rank approximation of original matrix
- So truncated, U_k (or V_k^T) is a new low dimensional representation of the word
- Typical values for *k* are 100-5000

$$U \qquad\qquad\qquad\qquad\qquad U_k$$

$$|V| \overbrace{\begin{bmatrix} 2.2 & 0.3 & & -2.4 & & 8.7 \\ 5.5 & -2.8 & \cdots & 1.1 & \cdots & 0.1 \\ -1.3 & 3.7 & & 4.7 & & 3.5 \\ \vdots & & \ddots & \vdots & \ddots & \vdots \\ 2.9 & -2.1 & \cdots & -3.3 & \cdots & -1.9 \end{bmatrix}}^{m \quad | \quad k} \Rightarrow |V| \overbrace{\begin{bmatrix} 2.2 & 0.3 & & -2.4 \\ 5.5 & -2.8 & \cdots & 1.1 \\ -1.3 & 3.7 & & 4.7 \\ & \vdots & \ddots & \vdots \\ 2.9 & -2.1 & \cdots & -3.3 \end{bmatrix}}^{k}$$

Word Vectors!

Words as Context

- Lists how often words appear with other words
  - In some predefined context (usually a window)
- The obvious problem with raw frequency: dominated by common words

	…	the	country	hell	…
…					
state		1973	10	1	
fun		54	2	0	
heaven		55	1	3	
……					

Pointwise Mutual Information

- For two events *x* and *y*, PMI computes the discrepancy between:
  - Their joint distribution
  - Their individual distributions (assuming independence)

$$PMI(x, y) = \log_2 \frac{p(x, y)}{p(x)p(y)}$$

Calculating PMI

	…	the	country	hell	…		Σ
…							
state		1973	10	1			12786
fun		54	2	0			633
heaven		55	1	3			627
…							
Σ		1047519	3617	780			15871304

$p(x,y) = count(x,y) / \Sigma$

$p(x) = \Sigma_x / \Sigma$

$p(y) = \Sigma_y / \Sigma$

x = *state*, y = *country*

$p(x,y) = 10/15871304 = 6.3 \times 10^{-7}$

$p(x) = 12786/15871304 = 8.0 \times 10^{-4}$

$p(y) = 3617/15871304 = 2.3 \times 10^{-4}$

$PMI(x,y) = \log_2(6.3 \times 10^{-7})/((8.0 \times 10^{-4})\ (2.3 \times 10^{-4}))$

$= 1.78$

PMI Matrix

- PMI does a better job of capturing interesting semantics
  - E.g. *heaven* and *hell*
- But it is obviously biased towards rare words
- And doesn't handle zeros well

	…	the	country	hell	…
…					
state		1.22	1.78	0.63	
fun		0.37	3.79	-inf	
heaven		0.41	2.80	6.60	
……					

PMI Tricks

- Zero all negative values (PPMI)
  - Avoid –inf and unreliable negative values
- Counter bias towards rare events
  - Smooth probabilities

SVD

	…	the	country	hell	…
…					
425		0	26.0	6.2	
426		0	5.2	0	
427		0	0	18.6	
…					

tf-idf matrix

	…	the	country	hell	…
…					
state		1.22	1.78	0.63	
fun		0.37	3.79	0	
heaven		0.41	2.80	6.60	
……					

PPMI matrix

- Regardless of whether we use document or word as context, SVD can be applied to create dense vectors

Similarity

- Word similarity = comparison between word vectors (e.g. cosine similarity)
- Find synonyms, based on proximity in vector space
  - automatic construction of lexical resources
- Use vectors as features in classifier — more robust to different inputs (*movie* vs *film*)

Neural Methods

Word Embeddings

- We’ve seen word embeddings used in neural networks (feedforward or recurrent)
- But these models are designed for other tasks:
  - Classification
  - Language modelling
- Word embeddings is just one part of the model

Neural Models for Embeddings

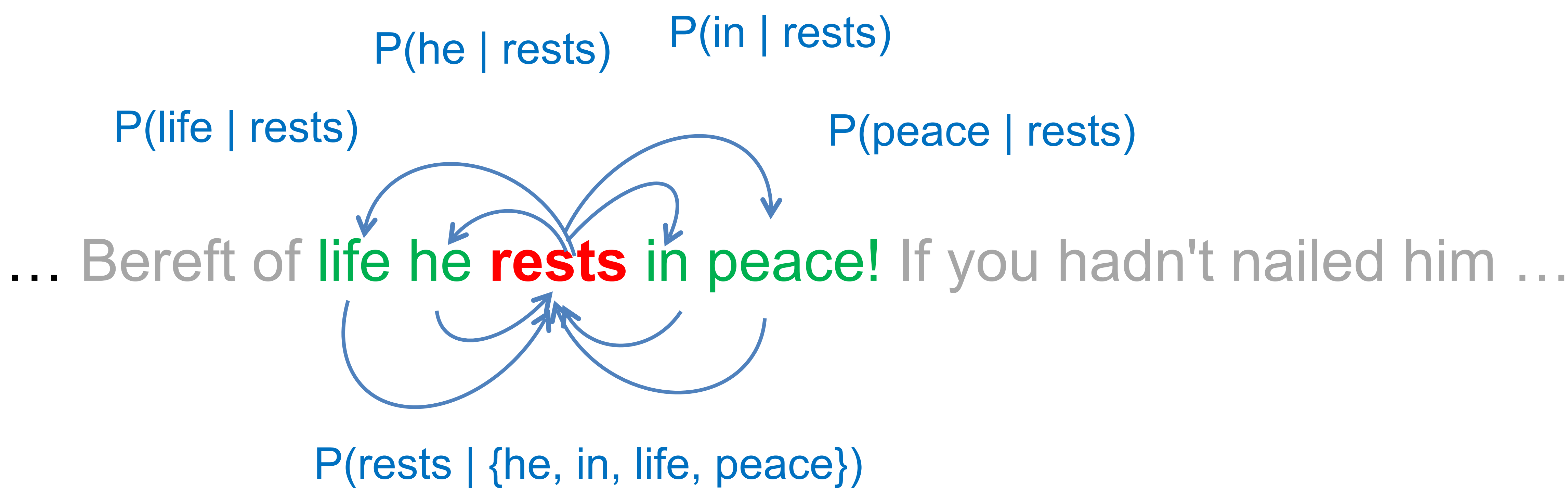
- Can we design neural networks whose goal is to learn word embeddings?
- Desiderata:
  - Unsupervised
  - Efficient
  - Useful representation

Word2Vec

- Neural network inspired approaches seek to learn vector representations of words and their contexts
- Key idea
  - *Word embeddings should be similar to embeddings of neighbouring words*
  - *And dissimilar to other words that don't occur nearby*
- Using vector dot product for vector 'comparison'
  - $u . v = \sum_j u_j v_j$

Word2Vec

- Framed as learning a classifier
  - **Skip-gram**: predict words in local context surrounding given word



  - **CBOW**: predict word in centre, given words in the local surrounding context
- Local context means words within L positions, L=2 above

Skip-gram Model

- Generates each word in context given centre word

P(he | rests) P(in | rests)

P(life | rests) P(peace | rests)

… Bereft of life he **rests** in peace! If you hadn't nailed him …

- Total probability defined as
  - Where subscript denotes position in running text

$$\prod_{l \in -L,\ldots,-1,1,\ldots,L} P(w_{t+l}|w_t)$$

- Using a logistic regression model

he rests

$$P(w_k|w_j) = \frac{\exp(c_{w_k} \cdot v_{w_j})}{\sum_{w' \in V} \exp(c_{w'} \cdot v_{w_j})}$$

Embedding parameterisation

- Two parameter matrices, with d-dimensional embedding for all words

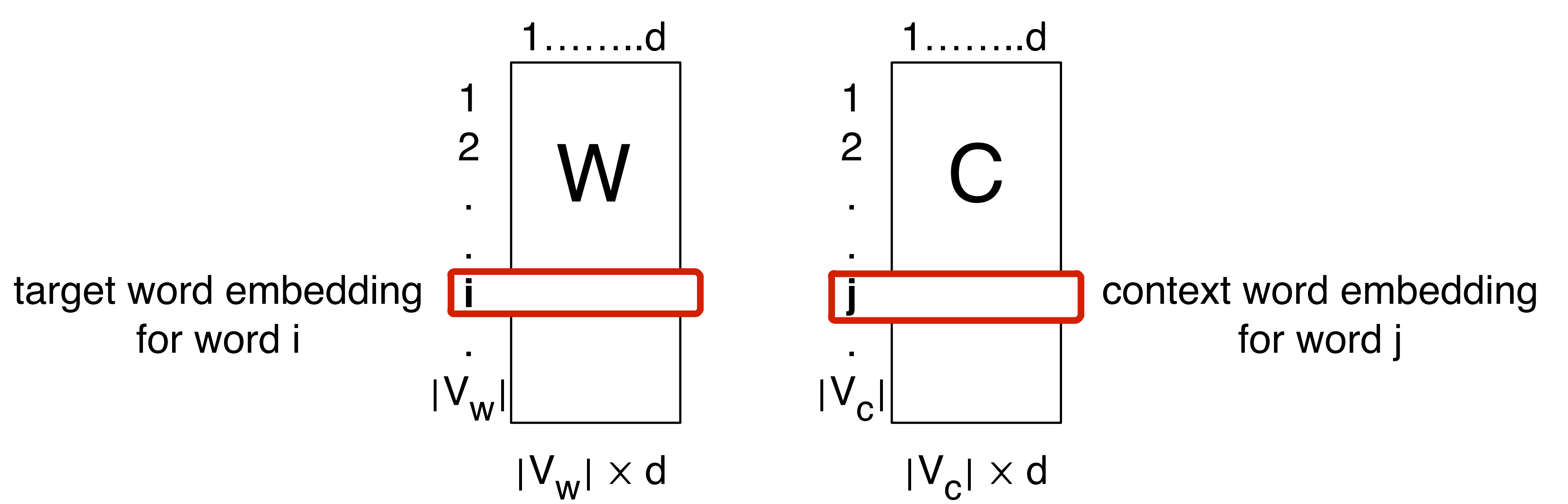


Fig 19.17, JM3

- Words are numbered, e.g., by sorting vocabulary and using word location as its index

Skip-gram model

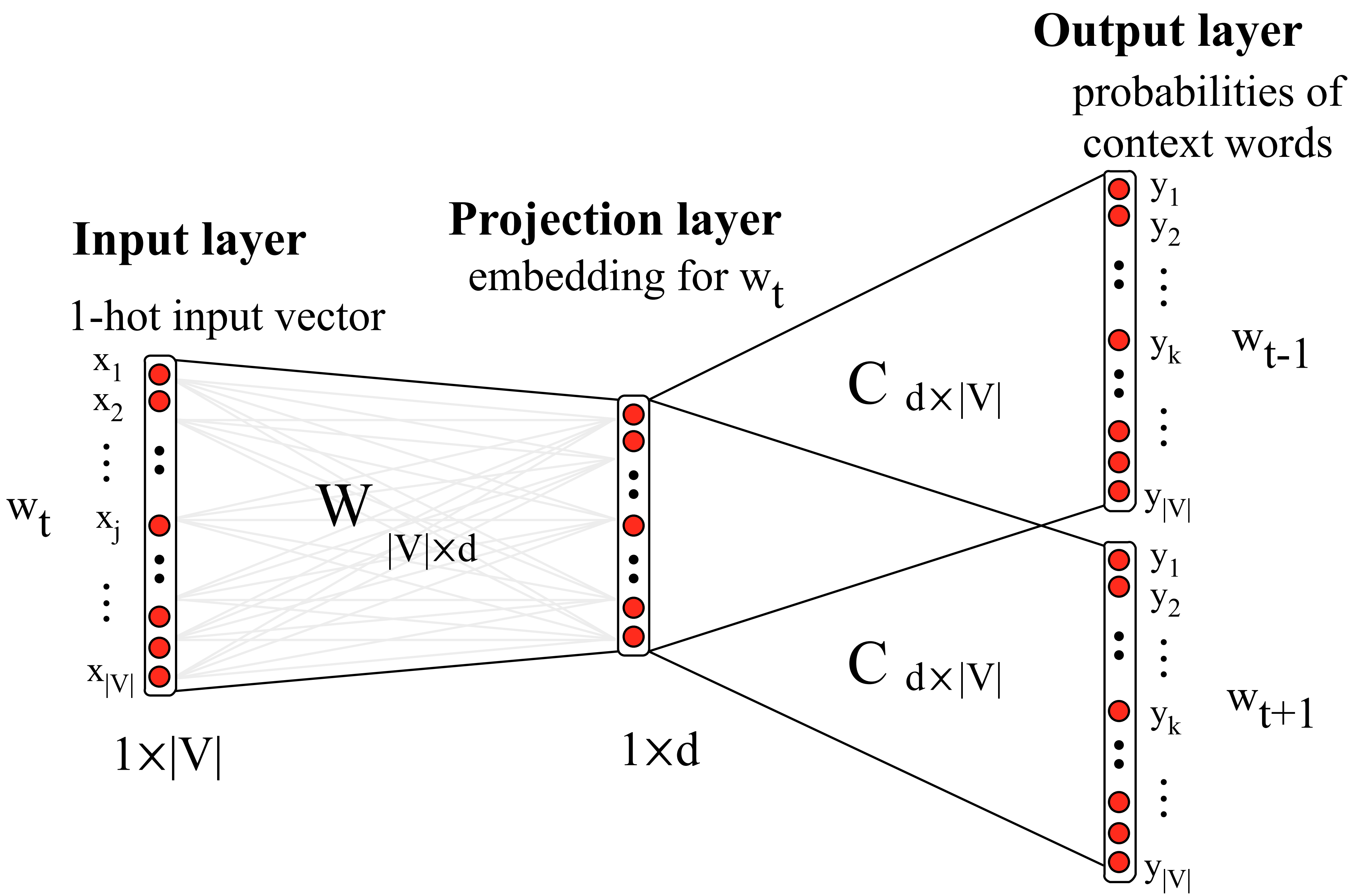


Fig 19.18, JM3

Training the skip-gram model

- Train to maximise likelihood of **raw text**
- Too slow in practice, due to normalisation over |V|

$$P(w_k|w_j) = \frac{\exp(c_{w_k} \cdot v_{w_j})}{\sum_{w' \in V} \exp(c_{w'} \cdot v_{w_j})}$$

- Reduce problem to binary classification, distinguish real context words from non-context words aka **"negative samples"**
  - words drawn randomly from V

Negative Sampling

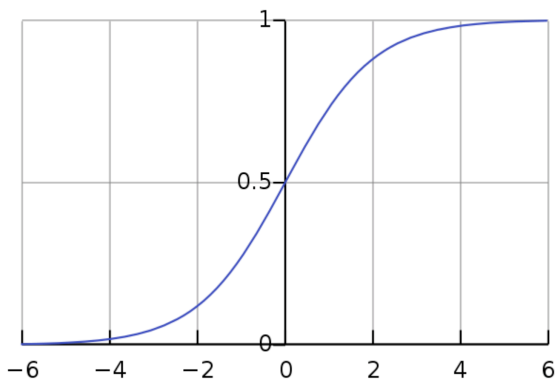
```
... lemon,  a [tablespoon of apricot jam,      a] pinch ...
               c1         c2     t      c3      c4
```

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



$$\frac{1}{1+e^{-x}}$$

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

maximise similarity between target word and real context words

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

minimise similarity between target word and non-context words

Skip-gram Loss

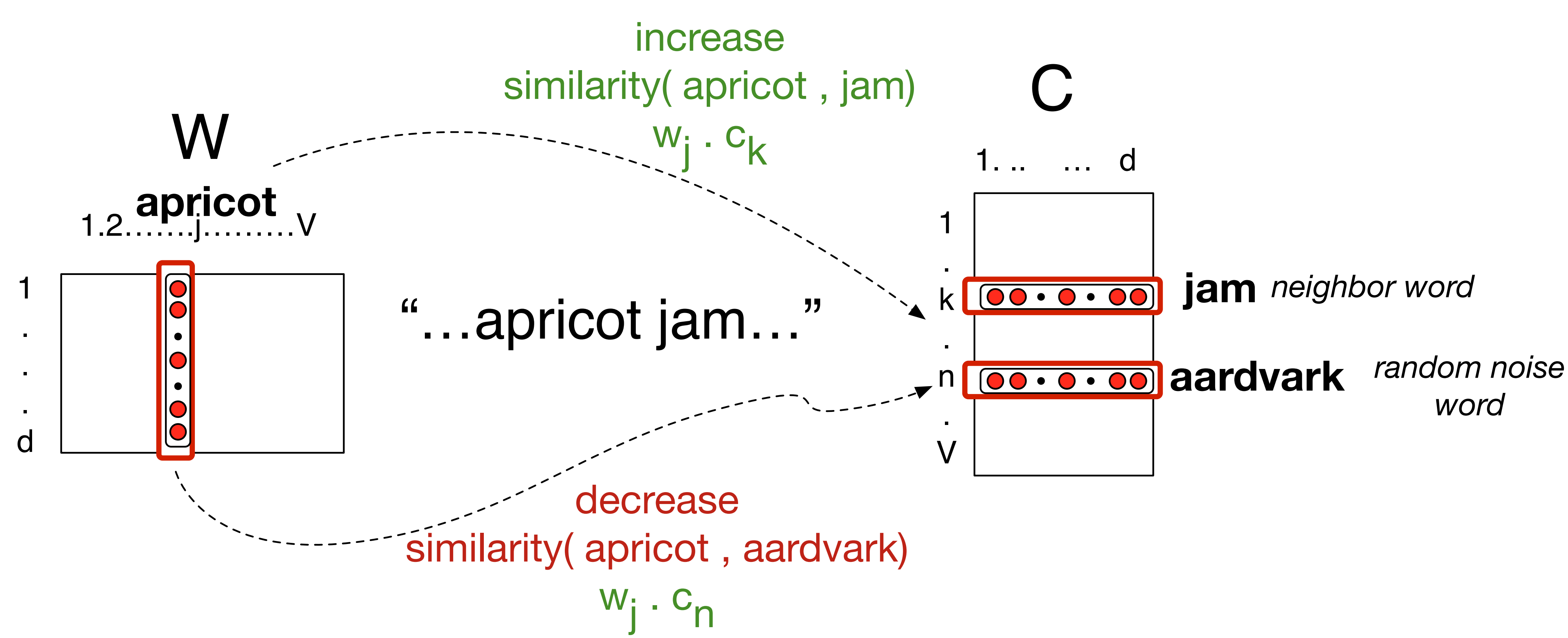
$$L(\theta) = \sum_{(t,c)\in +} \log P(+|t,c) + \sum_{(t,c)\in -} \log P(-|t,c)$$

- In practice, use *k* negative examples

$$L(\theta) \;=\; \log P(+|t,c) + \sum_{i=1}^{k} \log P(-|t,n_i)$$

Training Illustration

- Iterative process (stochastic gradient descent)
  - each step moves embeddings closer for context words
  - and moves embeddings apart for noise samples



Source: JM3 Ch 6

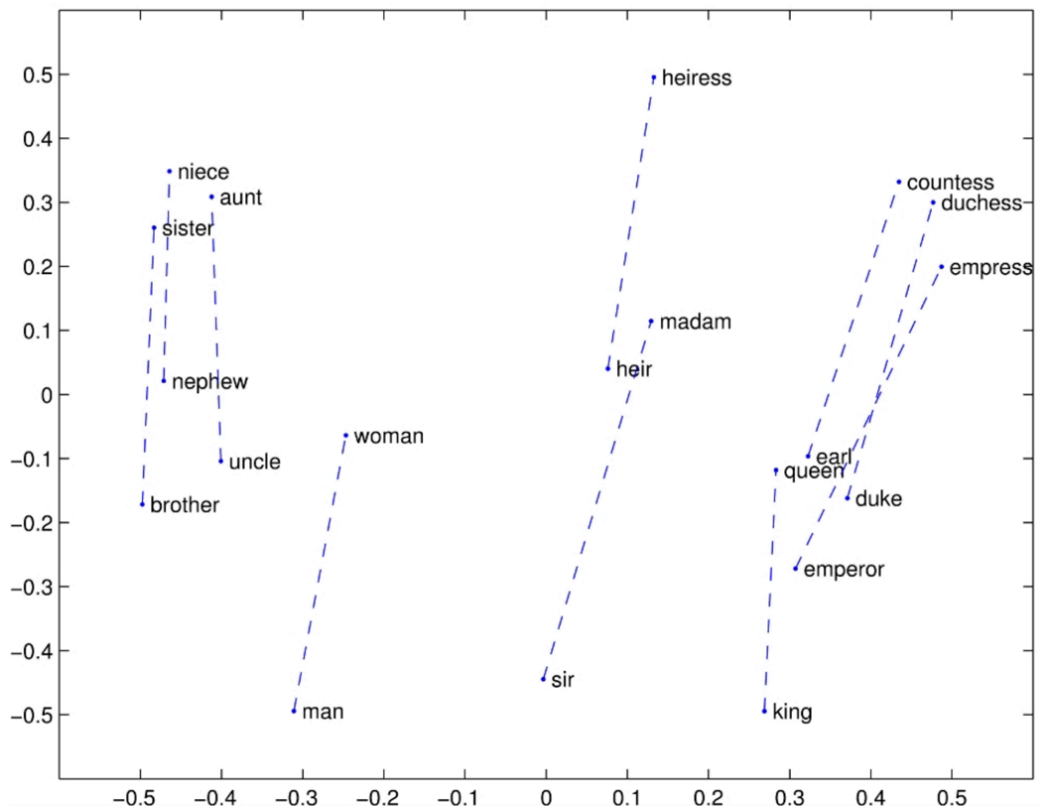
Desiderata

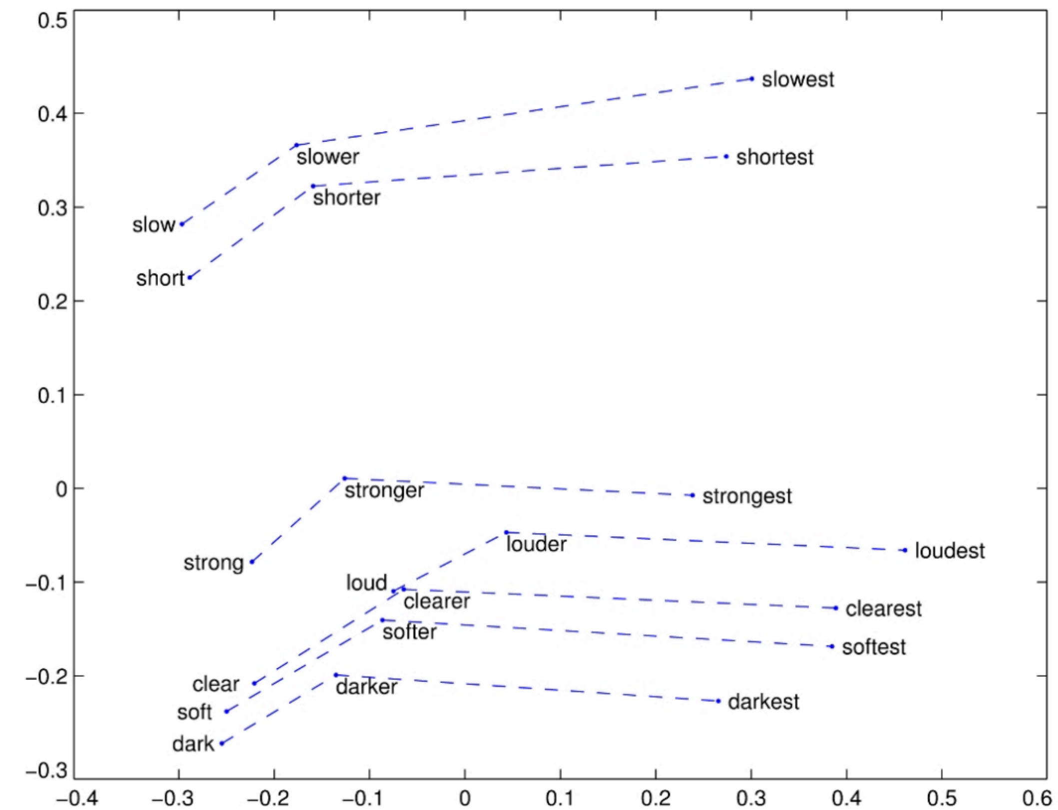
- Unsupervised
  - Raw, unlabelled corpus
- Efficient
  - Negative sampling (avoid softmax over full vocabulary)
  - Scales to very very large corpus
- Useful representation:
  - How do we evaluate word vectors?

Evaluating Word Vectors

- Lexicon style tasks
  - *WordSim-353* are pairs of nouns with judged relatedness
  - *SimLex-999* also covers verbs and adjectives
  - *TOEFL* asks for closest synonym as multiple choice
  - *…*
- Test compatibility of word pairs using cosine similarity in vector space

Embeddings Exhibit Meaningful Geometry





- Word analogy task
  - *Man* is to *King* as *Woman* is to ???
  - v(*Man*) - v(*King*) = v(*Woman*) - v(???)
  - v(???) = v(*Woman*) - v(*Man*) + v(*King*)

Evaluating Word Vectors

- Best evaluation is in other downstream tasks
  - Use bag-of-word embeddings as a feature representation in a classifier
  - First layer of most deep learning models is to embed input text; use pre-trained word vectors as embeddings
- Recently **contextual word vectors** shown to work even better
  - ELMO & BERT (next lecture!)

Pointers to Software

- Word2Vec
  - C implementation of Skip-gram and CBOW https://code.google.com/archive/p/word2vec/
- GenSim
  - Python library with many methods include LSI, topic models and Skip-gram/CBOW https://radimrehurek.com/gensim/index.html
- GLOVE
  - http://nlp.stanford.edu/projects/glove/

Further Reading

- JM3, Ch 6