CSE 447/547
Natural Language Processing
Winter 2018

Distributed Semantics & Embeddings

Yejin Choi - University of Washington

[Slides adapted from Dan Jurafsky]

Why vector models of meaning? computing the similarity between words

"**fast**" is similar to "**rapid**"

"**tall**" is similar to "**height**"

Question answering:

*Q: "How **tall** is Mt. Everest?"*
*Candidate A: "The official **height** of Mount Everest is 29029 feet"*

Similar words in plagiarism detection

MAINFRAMES

Mainframes are primarily referred to large computers with rapid, advanced processing capabilities that can execute and perform tasks equivalent to many Personal Computers (PCs) machines networked together. It is characterized with high quantity Random Access Memory (RAM), very large secondary storage devices, and high-speed processors to cater for the needs of the computers under its service.

Consisting of advanced components, mainframes have the capability of running multiple large applications required by many and most enterprises and organizations. This is one of its advantages. Mainframes are also suitable to cater for those applications (programs) or files that are of very high demand by its users (clients). Examples of such organizations and enterprises using mainframes are online shopping websites such as Ebay, Amazon, and computing-giant

MAINFRAMES

Mainframes usually are referred those computers with fast, advanced processing capabilities that could perform by itself tasks that may require a lot of Personal Computers (PC) Machines. Usually mainframes would have lots of RAMs, very large secondary storage devices, and very fast processors to cater for the needs of those computers under its service.

Due to the advanced components mainframes have, these computers have the capability of running multiple large applications required by most enterprises, which is one of its advantage. Mainframes are also suitable to cater for those applications or files that are of very large demand by its users (clients). Examples of these include the large online shopping websites -i.e. : Ebay, Amazon, Microsoft, etc.

Problems with thesaurus-based meaning

- We don't have a thesaurus for every language
- We can't have a thesaurus for every year
  - For historical linguistics, we need to compare word meanings in year t to year t+1
- Thesauruses have problems with **recall**
  - Many words and phrases are missing
  - Thesauri work less well for verbs, adjectives

Intuition of distributional word similarity

- Suppose I asked you what is ***tesgüino***?

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

- From context words humans can guess ***tesgüino*** means
  - an alcoholic beverage like beer
- Intuition for algorithm:
  - Two words are similar if they have similar word contexts.

Distributional models of meaning = vector-space models of meaning = vector semantics

Intuitions: Zellig Harris (1954):

- "oculist and eye-doctor … occur in almost the same environments"
- "If A and B have almost identical environments we say that they are synonyms."

Firth (1957):

- "You shall know a word by the company it keeps!"

Four kinds of vector models

Sparse vector representations

1. Word co-occurrence matrices

   -- weighted by mutual-information

Dense vector representations:

2. Singular value decomposition (and Latent Semantic Analysis)
3. Neural-network inspired models (skip-grams, CBOW)
4. Brown clusters

Shared intuition

- Model the meaning of a word by "embedding" it in a vector space.
- The meaning of a word is a vector of numbers
  - Vector models are also called "embeddings".

Vector Semantics

I. Words and co-occurrence vectors

Co-occurrence Matrices

- We represent how often a word occurs in a document
  - **Term-document matrix**
- Or how often a word occurs with another
  - **Term-term matrix**

    (or **word-word co-occurrence matrix**

    or **word-context matrix**)

Term-document matrix

- Each cell: count of word *w* in a document *d*:
  - Each document is a count vector in $\mathbb{N}^V$: a column below

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	1	8	15
soldier	2	2	12	36
fool	37	58	1	5
clown	6	117	0	0

Similarity in term-document matrices

Two documents are similar if their vectors are similar

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	1	8	15
soldier	2	2	12	36
fool	37	58	1	5
clown	6	117	0	0

The words in a term-document matrix

- Each word is a count vector in $\mathbb{N}^D$: a row below

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	1	8	15
soldier	2	2	12	36
fool	37	58	1	5
clown	6	117	0	0

The words in a term-document matrix

- Two **words** are similar if their vectors are similar

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	1	8	15
soldier	2	2	12	36
fool	37	58	1	5
clown	6	117	0	0

The word-word or word-context matrix

- Instead of entire documents, use smaller contexts
  - Paragraph
  - Window of ± 4 words
- A word is now defined by a vector over counts of context words
- Instead of each vector being of length D
- Each vector is now of length |V|
- The word-word matrix is |V|x|V|

Word-Word matrix
Sample contexts ± 7 words

sugar, a sliced lemon, a tablespoonful of	**apricot**	preserve or jam, a pinch each of,
their enjoyment. Cautiously she sampled her first	**pineapple**	and another fruit whose taste she likened
well suited to programming on the digital	**computer**.	In finding the optimal R-stage policy from
for the purpose of gathering data and	**information**	necessary for the study authorized in the

	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	
…	…						

Word-word matrix

- We showed only 4x6, but the real matrix is 50,000 x 50,000
  - So it's very **sparse** (most values are 0)
  - That's OK, since there are lots of efficient algorithms for sparse matrices.
- The size of windows depends on your goals
  - The shorter the windows…
    - the more **syntactic** the representation ($\pm$ 1-3 words)
  - The longer the windows…
    - the more **semantic** the representation ($\pm$ 4-10 words)

Vector Semantics

Positive Pointwise Mutual Information (PPMI)

Informativeness of a context word X for a target word Y

- Freq(the, beer)   VS   freq(drink, beer) ?
- How about joint probability?
- P(the, beer)   VS   (drink, beer) ?
- Frequent words like "the" and "of" are not quite informative
- Normalize by the individual word frequencies!
  ➔ Pointwise Mutual Information (PMI)

Pointwise Mutual Information

Pointwise mutual information:

Do events x and y co-occur more than if they were independent?

$$PMI(X = x, Y = y) = \log_2 \frac{P(x, y)}{P(x)P(y)}$$

PMI between two words: (Church & Hanks 1989)

Do words x and y co-occur more than if they were independent?

$$\text{PMI}(word_1, word_2) = \log_2 \frac{P(word_1, word_2)}{P(word_1)P(word_2)}$$

Positive Pointwise Mutual Information

- PMI ranges from $-\infty$ to $+\infty$
- But the negative values are problematic
  - Things are co-occurring **less than** we expect by chance
  - Unreliable without enormous corpora
    - Imagine w1 and w2 whose probability is each 10^{-6}
    - Hard to be sure p(w1,w2) is significantly different than 10^{-12}
  - Plus it's not clear people are good at "unrelatedness"
- So we just replace negative PMI values by 0

- Positive PMI (PPMI) between word1 and word2:

$$\text{PPMI}(word_1, word_2) = \max\left(\log_2 \frac{P(word_1, word_2)}{P(word_1)P(word_2)}, 0\right)$$

Computing PPMI on a term-context matrix

- Matrix $\mathbf{F}$ with $\mathbf{W}$ rows (words) and $\mathbf{C}$ columns (contexts)
- f_{ij} is # of times w_i occurs in context c_j

	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

$$p_{ij} = \frac{f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}}$$

$$p_{i*} = \frac{\sum_{j=1}^{C} f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}}$$

$$p_{*j} = \frac{\sum_{i=1}^{W} f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}}$$

$$\mathrm{PMI}_{ij} = \log \frac{p_{ij}}{p_{i*}p_{*j}}$$

$$\mathrm{PPMI}_{ij} = \max(0, \mathrm{PMI}_{ij})$$

$$p_{ij} = \frac{f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}}$$

Count(w,context)

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

$$p(w_i) = \frac{\sum_{j=1}^{C} f_{ij}}{N}$$

p(w=information,c=data) = 6/19 = .32

p(w=information) = 11/19 = .58

p(c=data) = 7/19 = .37

p(w,context) / **p(w)**

	computer	data	pinch	result	sugar	p(w)
apricot	0.00	0.00	0.05	0.00	0.05	0.11
pineapple	0.00	0.00	0.05	0.00	0.05	0.11
digital	0.11	0.05	0.00	0.05	0.00	0.21
information	0.05	0.32	0.00	0.21	0.00	0.58
p(context)	0.16	0.37	0.11	0.26	0.11	

$$\mathrm{PMI}_{ij} = \log \frac{p_{ij}}{p_{i*}p_{*j}}$$

	p(w,context)					**p(w)**
	computer	data	pinch	result	sugar	
apricot	0.00	0.00	0.05	0.00	0.05	0.11
pineapple	0.00	0.00	0.05	0.00	0.05	0.11
digital	0.11	0.05	0.00	0.05	0.00	0.21
information	0.05	0.32	0.00	0.21	0.00	0.58
p(context)	0.16	0.37	0.11	0.26	0.11	

- pmi(information,data) = $\log_2$ (.32 / (.37*.58)) = .58

(.57 using full precision)

PPMI(w,context)

	computer	data	pinch	result	sugar
apricot	-	-	2.25	-	2.25
pineapple	-	-	2.25	-	2.25
digital	1.66	0.00	-	0.00	-
information	0.00	0.57	-	0.47	-

Weighting PMI

- PMI is biased toward infrequent events
  - Very rare words have very high PMI values
- Two solutions:
  - Give rare words slightly higher probabilities
  - Use add-one smoothing (which has a similar effect)

Weighting PMI: Giving rare context words slightly higher probability

- Raise the context probabilities to $\alpha = 0.75$:

$$\text{PPMI}_\alpha(w,c) = \max(\log_2 \frac{P(w,c)}{P(w)P_\alpha(c)}, 0)$$

$$P_\alpha(c) = \frac{count(c)^\alpha}{\sum_c count(c)^\alpha}$$

- This helps because $P_\alpha(c) > P(c)$ for rare c
- Consider two events, P(a) = .99 and P(b)=.01
- $P_\alpha(a) = \frac{.99^{.75}}{.99^{.75}+.01^{.75}} = .97 \quad P_\alpha(b) = \frac{.01^{.75}}{.99^{.75}+.01^{.75}} = .03$

TF-IDF: Alternative to PPMI for measuring association

- tf-idf (that's a hyphen not a minus sign)
- The combination of two factors
  - Term frequency (Luhn 1957): frequency of the word
  - Inverse document frequency (IDF) (Sparck Jones 1972)
    - N is the total number of documents
    - df_i = "document frequency of word *i*"
      = # of documents with word *I*

$$\text{idf}_i = \log\left(\frac{N}{df_i}\right)$$

- $w_{ij} = \text{tf}_{ij}\text{idf}_i$ = *weight of word* i *in document* j

Vector Semantics

Measuring similarity: the cosine

Measuring similarity

- Given 2 target words *v* and *w*
- We'll need a way to measure their similarity.
- Most measure of vectors similarity are based on:

- **Dot product** or **inner product** from linear algebra

$$\text{dot-product}(\vec{v},\vec{w}) = \vec{v}\cdot\vec{w} = \sum_{i=1}^{N} v_i w_i = v_1 w_1 + v_2 w_2 + \ldots + v_N w_N$$

  - High when two vectors have large values in same dimensions.
  - Low (in fact 0) for **orthogonal vectors** with zeros in complementary distribution

Problem with dot product

$$\text{dot-product}(\vec{v},\vec{w}) = \vec{v}\cdot\vec{w} = \sum_{i=1}^{N} v_i w_i = v_1 w_1 + v_2 w_2 + \ldots + v_N w_N$$

- Dot product is longer if the vector is longer. Vector length:

$$|\vec{v}| = \sqrt{\sum_{i=1}^{N} v_i^2}$$

- Vectors are longer if they have higher values in each dimension
- That means more frequent words will have higher dot products
- That's bad: we don't want a similarity metric to be sensitive to word frequency

Solution: cosine

- Just divide the dot product by the length of the two vectors!

$$\frac{\vec{a} \cdot \vec{b}}{|\vec{a}||\vec{b}|}$$

- This turns out to be the cosine of the angle between them!

$$\vec{a} \cdot \vec{b} = |\vec{a}||\vec{b}| \cos \theta$$

$$\frac{\vec{a} \cdot \vec{b}}{|\vec{a}||\vec{b}|} = \cos \theta$$

Cosine for computing similarity

Sec. 6.3

Dot product

Unit vectors

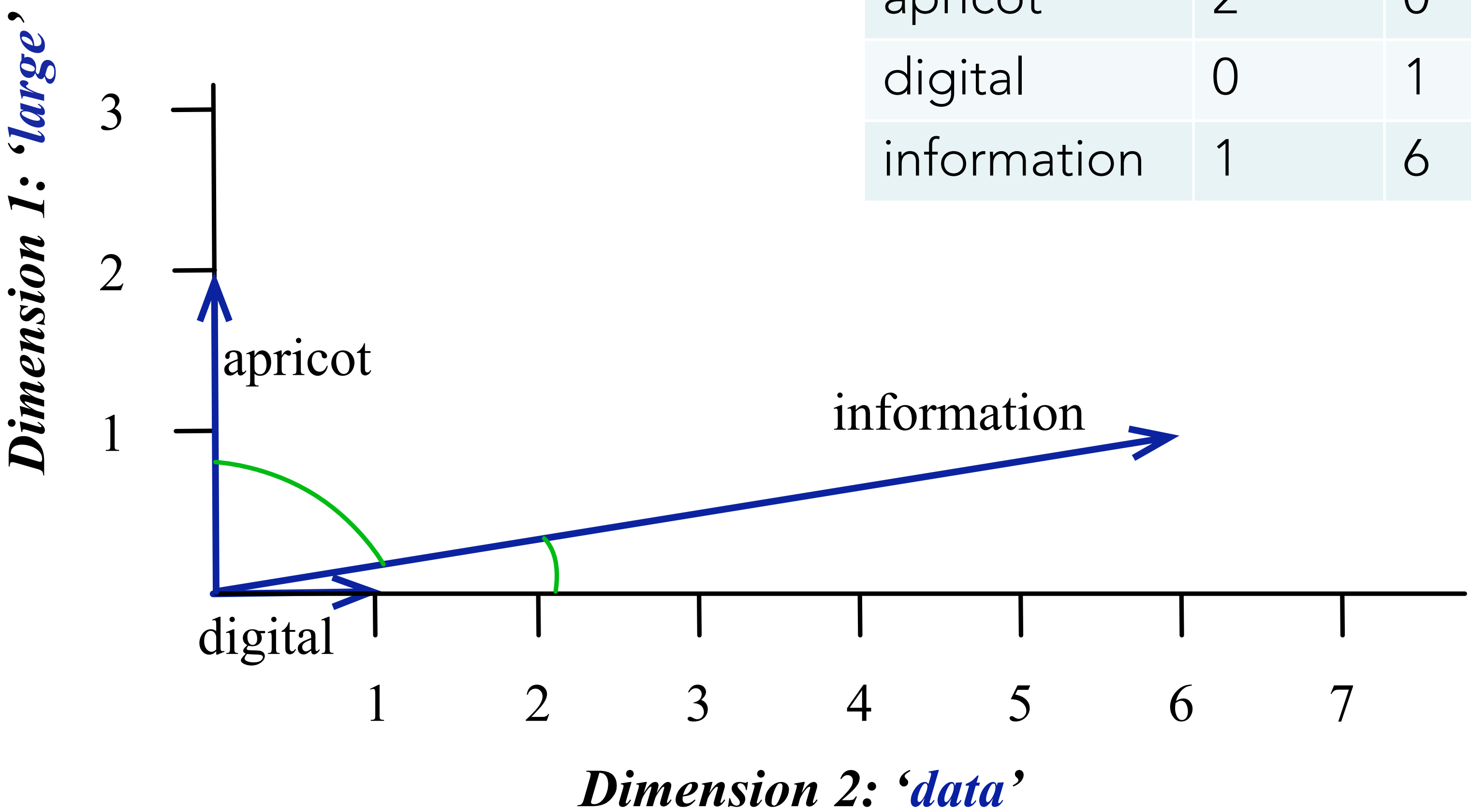
$$\cos(\vec{v},\vec{w}) = \frac{\vec{v}\bullet\vec{w}}{|\vec{v}||\vec{w}|} = \frac{\vec{v}}{|\vec{v}|}\bullet\frac{\vec{w}}{|\vec{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

v_i is the PPMI value for word v in context i
w_i is the PPMI value for word w in context i.

$\mathrm{Cos}(\vec{v},\vec{w})$ is the cosine similarity of $\vec{v}$ and $\vec{w}$

Visualizing vectors and angles

	large	data
apricot	2	0
digital	0	1
information	1	6



Vector Semantics

Dense Vectors

Sparse versus dense vectors

- PPMI vectors are
  - **long** (length |V|= 20,000 to 50,000)
  - **sparse** (most elements are zero)
- Alternative: learn vectors which are
  - **short** (length 200-1000)
  - **dense** (most elements are non-zero)

Sparse versus dense vectors

- Why dense vectors?
  - Short vectors may be easier to use as features in machine learning (less weights to tune)
  - Dense vectors may generalize better than storing explicit counts
  - They may do better at capturing synonymy:
    - *car* and *automobile* are synonyms; but are represented as distinct dimensions; this fails to capture similarity between a word with *car* as a neighbor and a word with *automobile* as a neighbor

Three methods for short dense vectors

- Singular Value Decomposition (SVD)
  - A special case of this is called LSA (Latent Semantic Analysis)
  - (See supplementary topics)
- Embeddings
  - skip-grams and CBOW
- Brown clustering

Vector Semantics

Embeddings inspired by neural language models: skip-grams and CBOW

Prediction-based models: An alternative way to get dense vectors

- **Skip-gram** (Mikolov et al. 2013a) **CBOW** (Mikolov et al. 2013b)
- Learn embeddings as part of the process of word prediction.
- Train a neural network to predict neighboring words
  - Inspired by **neural net language models** (sans nonlinearity).
  - In so doing, learn dense embeddings for the words in the training corpus.
- Advantages:
  - Fast, easy to train (much faster than SVD)
  - Available online in the `word2vec` package
  - Including sets of pretrained embeddings!

Skip-grams

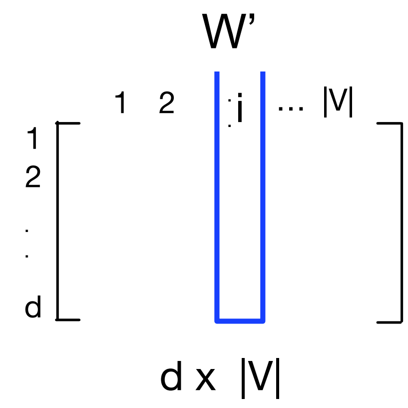
- Predict each neighboring word
  - in a context window of 2*C* words
  - from the current word.
- So for C=2, we are given word w_t and predicting these 4 words:

$$\left[w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}\right]$$

Skip-grams learn 2 embeddings for each w

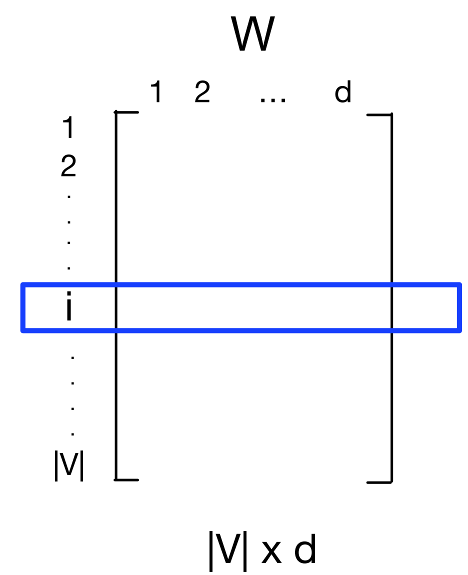
output embedding *v′*, in the output matrix W′

- Embedding of the context word
- Column *i* of the output matrix *W′* is a 1 x d embedding v'_i for word *i* in the vocabulary.



input embedding *v*, in the input matrix *W*

- Embedding of the target word
- Row *i* of the input matrix *W* is the *d x 1* embedding v_i for word *i* in the vocabulary



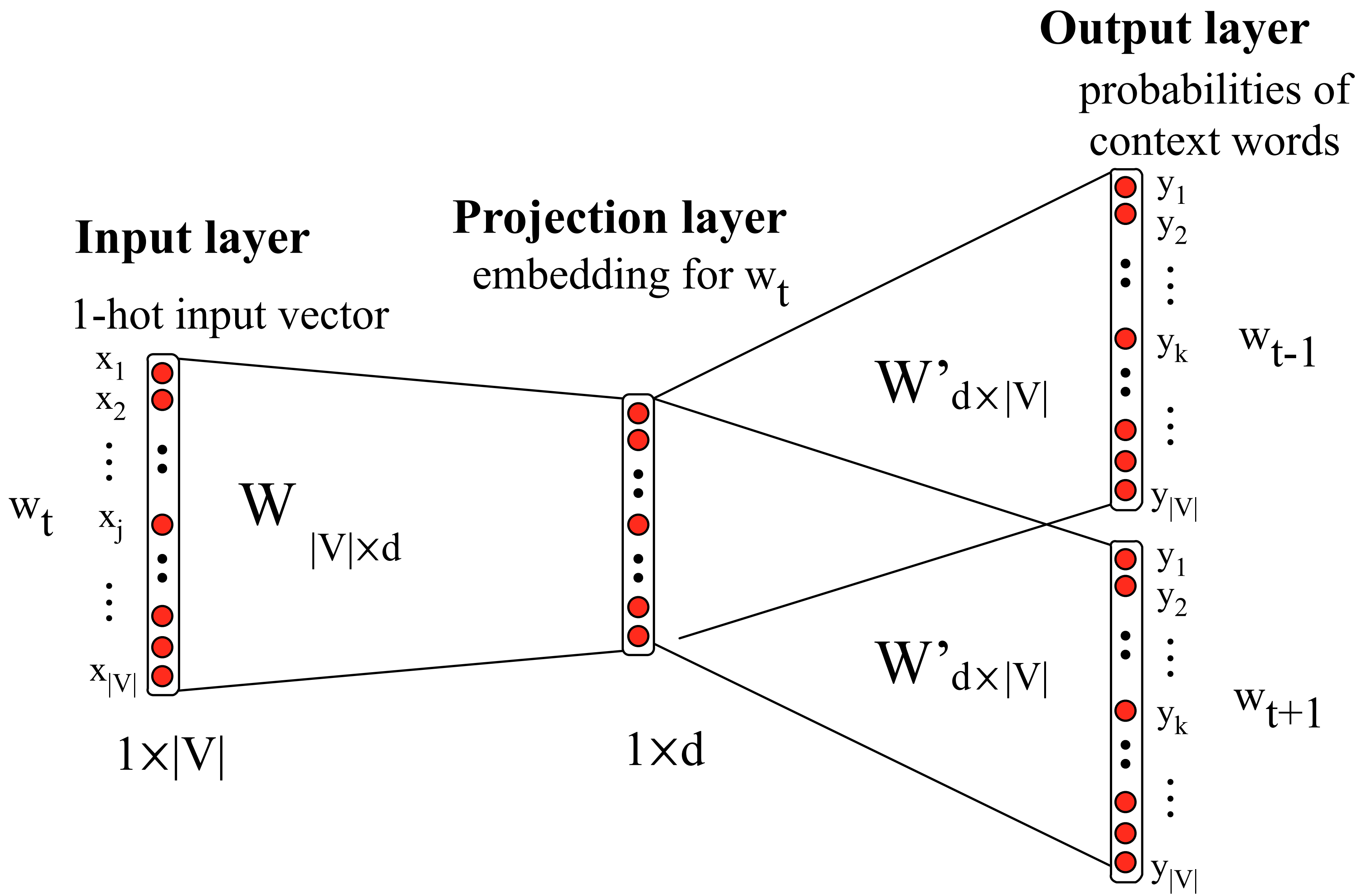
Setup

- Walking through corpus pointing at word $w(t)$, whose index in the vocabulary is j, so we'll call it w_j $(1 < j < |V|)$.
- Let's predict $w(t+1)$, whose index in the vocabulary is k $(1 < k < |V|)$. Hence our task is to compute $P(w_k|w_j)$.

One-hot vectors

- A vector of length |V|
- 1 for the target word and 0 for other words
- So if "popsicle" is vocabulary word 5
- The **one-hot vector** is
- [0,0,0,0,1,0,0,0,0.......0]

Skip-gram

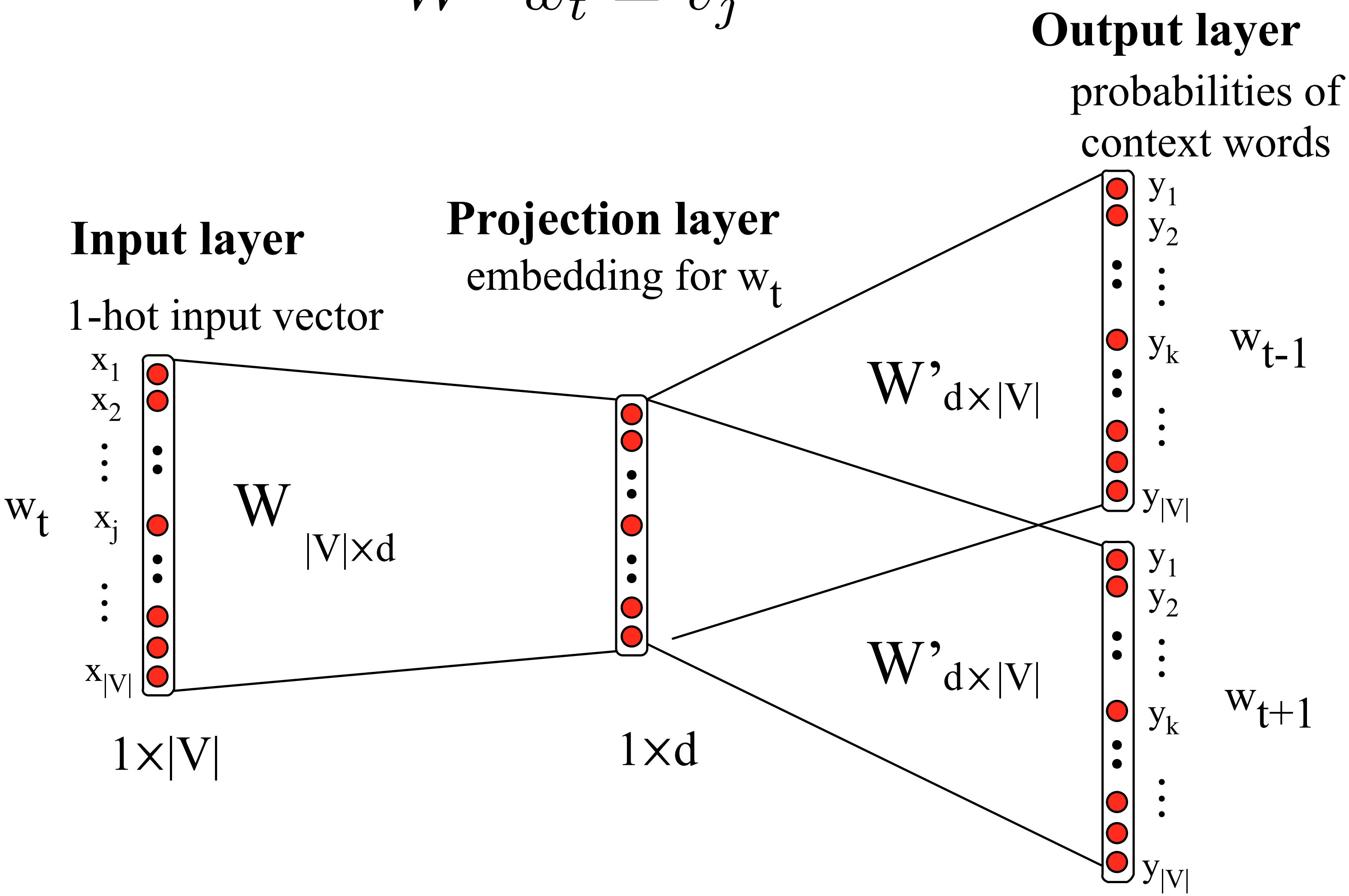

Output layer
probabilities of
context words
Projection layer
embedding for w t
Input layer
1-hot input vector
x 1
x 2
x j
x |V|
w t
W
|V|×d
W' d×|V|
W' d×|V|
y 1
y 2
y k
y |V|
w t-1
w t+1
1×|V|
1×d

Skip-gram

$$W'^T v_j = w_{t-1}$$

$$y_k = v'^T_k v_j$$

$$W^T w_t = v_j$$



Turning outputs into probabilities

$$y_k = v_k'^T v_j = v_k' \cdot v_j$$

- We use softmax to turn into probabilities

$$p(w_k|w_j) = \frac{exp(v_k' \cdot v_j)}{\sum_{w' \in |V|} exp(v_w' \cdot v_j)}$$

Embeddings from W and W'

- Since we have two embeddings, v_j and v'_j for each word w_j
- We can either:
  - Just use v_j
  - Sum them
  - Concatenate them to make a double-length embedding

Training embeddings

$$\underset{\theta}{\operatorname{argmax}} \ \log \ p(\text{Text})$$

$$\underset{\theta}{\operatorname{argmax}} \log \prod_{t=1}^{T} p(w^{(t-C)}, ..., w^{(t-1)}, w^{(t+1)}, ..., w^{(t+C)} | w^{(t)})$$

$$= \underset{\theta}{\operatorname{argmax}} \sum_{t=1}^{T} \sum_{-c \leq j \leq c, j \neq 0} \log p(w^{(t+j)} | w^{(t)})$$

$$= \underset{\theta}{\operatorname{argmax}} \sum_{t=1}^{T} \sum_{-c \leq j \leq c, j \neq 0} \log \frac{exp(v'^{(t+j)} \cdot v^{(t)})}{\sum_{w \in |V|} exp(v'_w \cdot v^{(t)})}$$

$$= \underset{\theta}{\operatorname{argmax}} \sum_{t=1}^{T} \sum_{-c \leq j \leq c, j \neq 0} \left[v'^{(t+j)} \cdot v^{(t)} - \log \sum_{w \in |V|} exp(v'_w \cdot v^{(t)}) \right]$$

Training: Noise Contrastive Estimation (NCE)

$$\underset{\theta}{\operatorname{argmax}} \ \log \ p(\text{Text})$$

$$= \underset{\theta}{\operatorname{argmax}} \sum_{t=1}^{T} \sum_{-c \le j \le c, j \ne 0} \left[v'^{(t+j)} \cdot v^{(t)} - \log \sum_{w \in |V|} exp(v'_w \cdot v^{(t)}) \right]$$

- the normalization factor is too expensive to compute exactly (why?)
- Negative sampling: sample only a handful of negative examples to compute the normalization factor
- (some engineering detail) the actual skip-gram training also converts the problem into binary classification (logistic regression) of predicting whether a given word is a context word or not

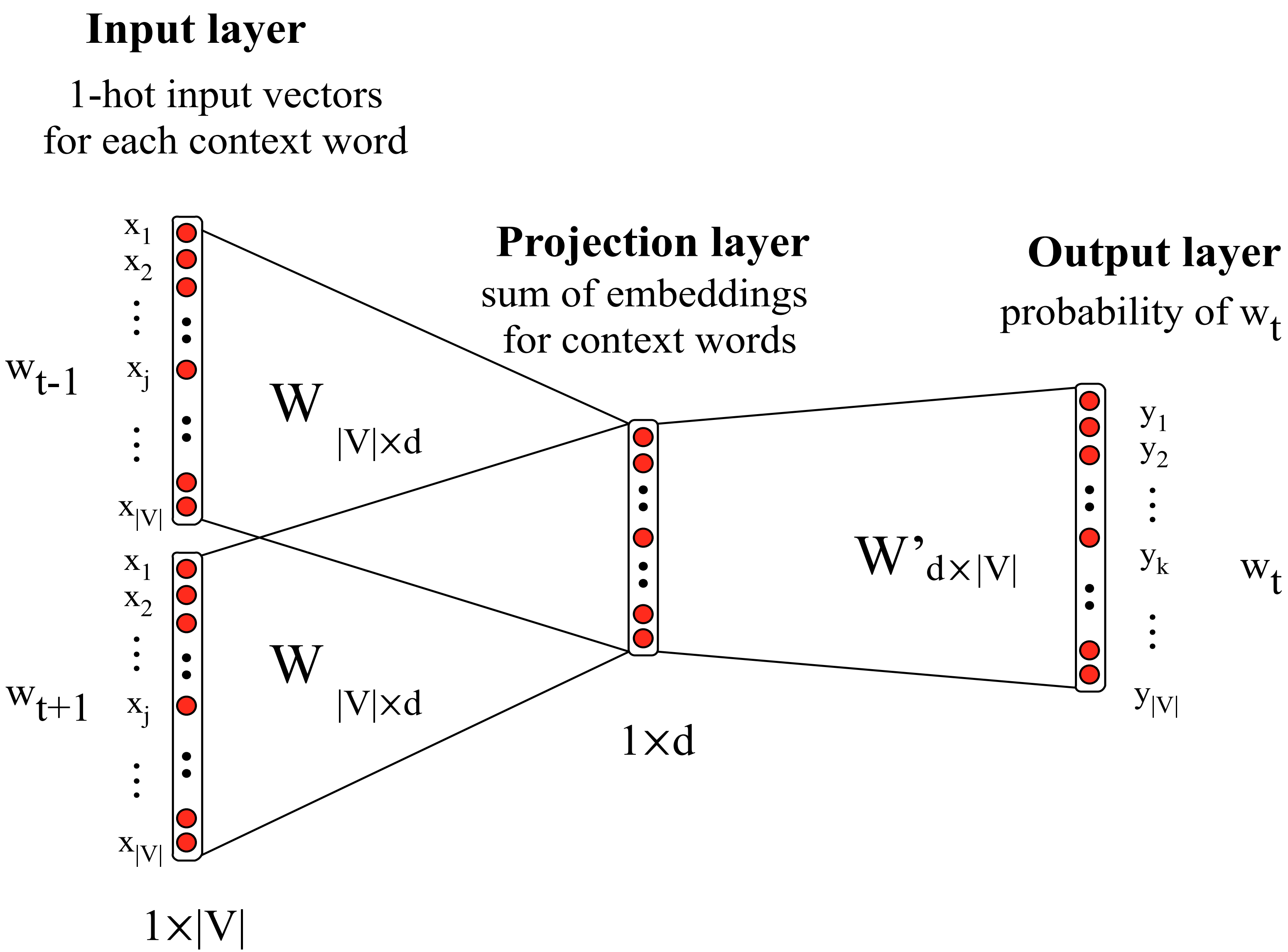
Relation between skipgrams and PMI!

- If we multiply WW'
- We get a |V|x|V| matrix M , each entry m_{ij} corresponding to some association between input word i and output word j
- Levy and Goldberg (2014b) show that skip-gram reaches its optimum just when this matrix is a shifted version of PMI:

$$WW' = M^{\text{PMI}} - \log k$$

- So skip-gram is implicitly factoring a shifted version of the PMI matrix into the two embedding matrices.

CBOW (Continuous Bag of Words)



Properties of embeddings

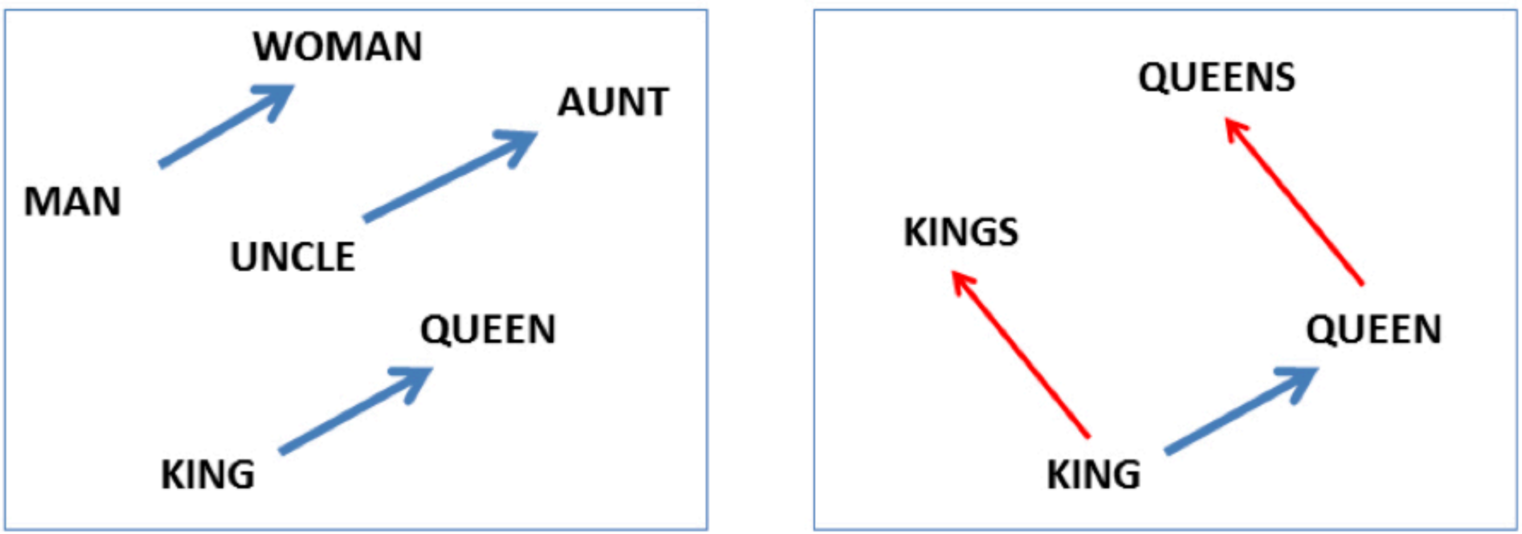
- Nearest words to some embeddings (Mikolov et al. 2013)

target:	Redmond	Havel	ninjutsu	graffiti	capitulate
	Redmond Wash.	Vaclav Havel	ninja	spray paint	capitulation
	Redmond Washington	president Vaclav Havel	martial arts	grafitti	capitulated
	Microsoft	Velvet Revolution	swordsmanship	taggers	capitulating

Embeddings capture relational meaning!

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

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



Vector Semantics

Brown clustering

Brown clustering

- An agglomerative clustering algorithm that clusters words based on which words precede or follow them
- These word clusters can be turned into a kind of vector
- We'll give a very brief sketch here.

Class-based language model

- Suppose each word was in some class $c_{i:}$

$$P(w_i|w_{i-1}) = P(c_i|c_{i-1})P(w_i|c_i)$$

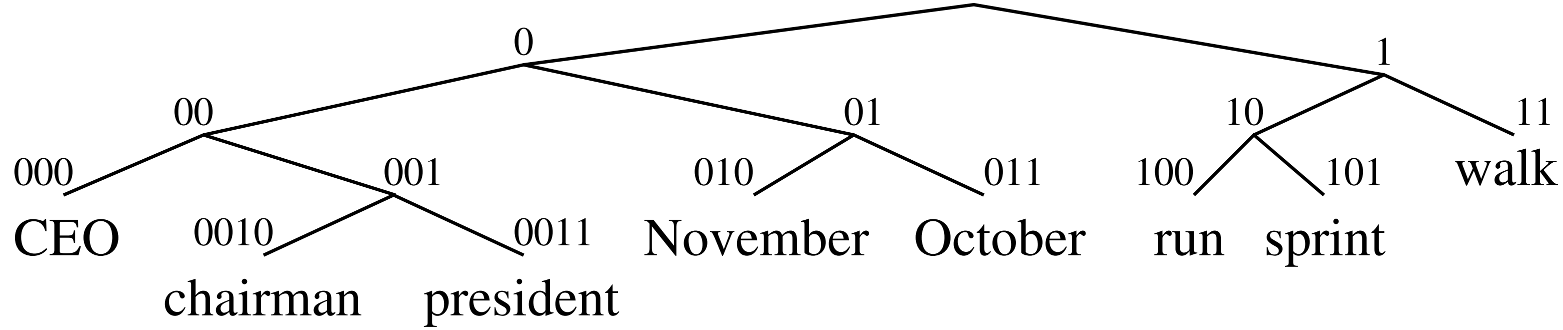
$$P(\text{corpus}|C) = \prod_{i-1}^{n} P(c_i|c_{i-1})P(w_i|c_i)$$

Brown clustering algorithm

- Each word is initially assigned to its own cluster.
- We now consider merging each pair of clusters. Highest quality merge is chosen.
  - Quality = merges two words that have similar probabilities of preceding and following words
  - (More technically quality = smallest decrease in the likelihood of the corpus according to a class-based language model)
- Clustering proceeds until all words are in one big cluster.

Brown Clusters as vectors

- By tracing the order in which clusters are merged, the model builds a binary tree from bottom to top.
- Each word represented by binary string = path from root to leaf
- Each intermediate node is a cluster
- Chairman is 0010, “months” = 01, and verbs = 1



Brown cluster examples

Friday Monday Thursday Wednesday Tuesday Saturday Sunday weekends Sundays Saturdays
June March July April January December October November September August
pressure temperature permeability density porosity stress velocity viscosity gravity tension
anyone someone anybody somebody
had hadn’t hath would’ve could’ve should’ve must’ve might’ve
asking telling wondering instructing informing kidding reminding bothering thanking deposing
mother wife father son husband brother daughter sister boss uncle
great big vast sudden mere sheer gigantic lifelong scant colossal
down backwards ashore sideways southward northward overboard aloft downwards adrift

Brown Clustering on Twitter!

http://www.cs.cmu.edu/~ark/TweetNLP/cluster_viewer.html

1110101010010 (52)
sorry gutted sry srry soz #thankful sory sorrry sowwy sori thankgod sorryy sowi sorri sorryyy sorrrry luckyyy sowwie paiseh sowie soory sorry- sorrrrry sowee -sorry sorryyyy #didntwannatellyou sorreh sorr sowy soorry sorrryyy apols sawry #iforgiveyou sryy sorrie sowwwy offski s0rry sorrryy soryy sorrrrrry sawwy sorryyyyy sozz nitm sowry thankgoodness sowwi

00101110 (79)
really rly realy genuinely rlly reallly realllly reallyy rele realli relly reallllly reali sholl rily reallyyy reeeeally realllllly reaally reeeally rili reaaally reaaaally reallyyyy rilly reallllllly reeeeeally reeally shol realllyyy reely re reaaaaally shole really2 reallyyyyy _really_ realllllllly reaaly realllyy realli reallt genuinly relli reallllyyyy reeeeeeally weally reaaallly reallllyyy

1110101100111 (582)
:(:/ -_- -.- :-(:'(d: :| :s -__- =(=/ >.< -___- :-/ </3 :\ -____- ;(/: ☹ :((>_< =[:[#fml 😒 -_____- =\ >:(😔 -,- >.> >:o ;/ 😳 d; .-. -______- >_> :(((-_-" =s ;_; #ugh :-\ =.= -_______-

Summary

- Distributional (vector) models of meaning
  - **Sparse** (PPMI-weighted word-word co-occurrence matrices)
  - **Dense**:
    - Word-word SVD 50-2000 dimensions
    - Skip-grams and CBOW
    - Brown clusters 5-20 binary dimensions.

Supplementary Topics

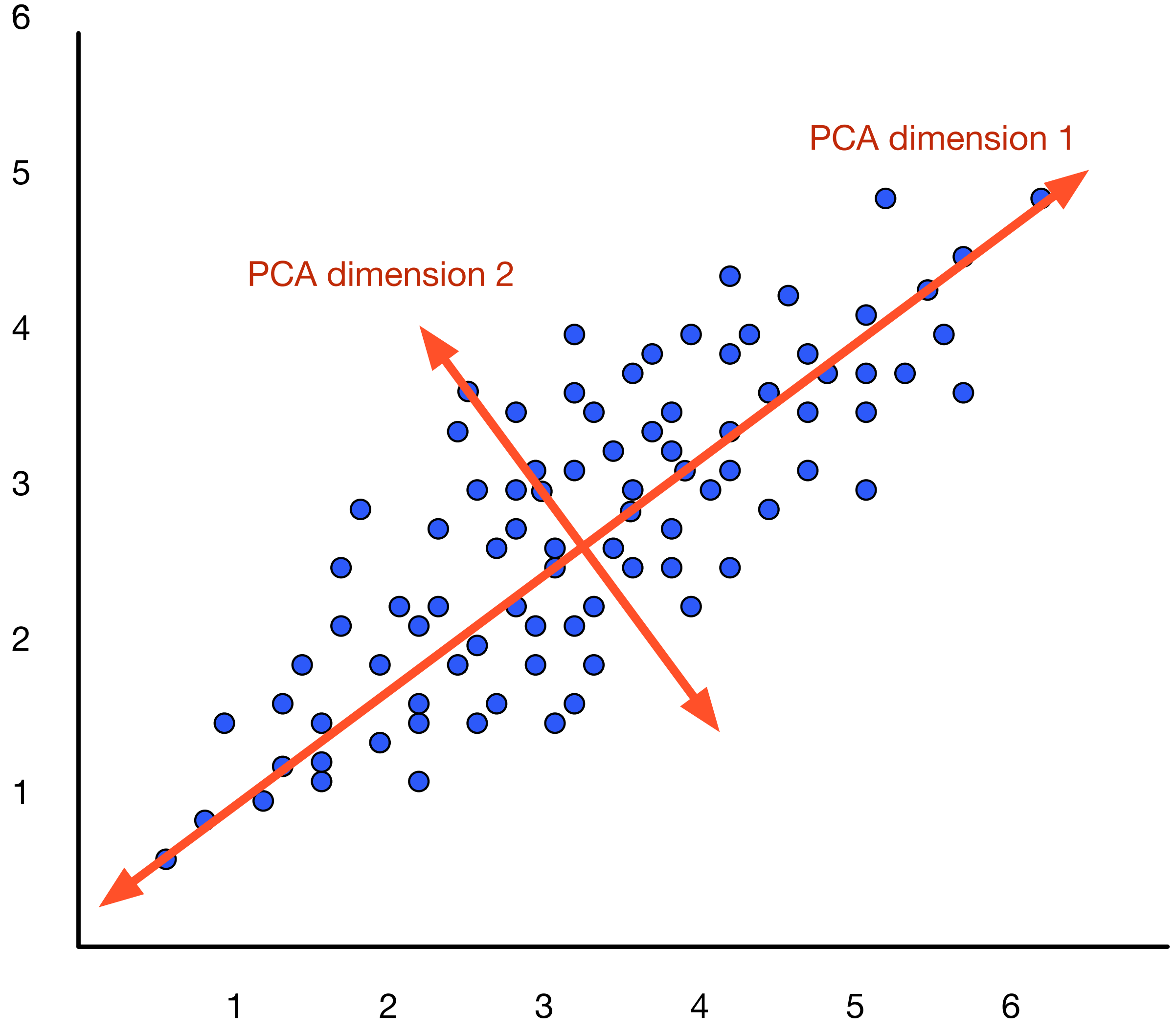
Vector Semantics

Dense Vectors via SVD

Intuition

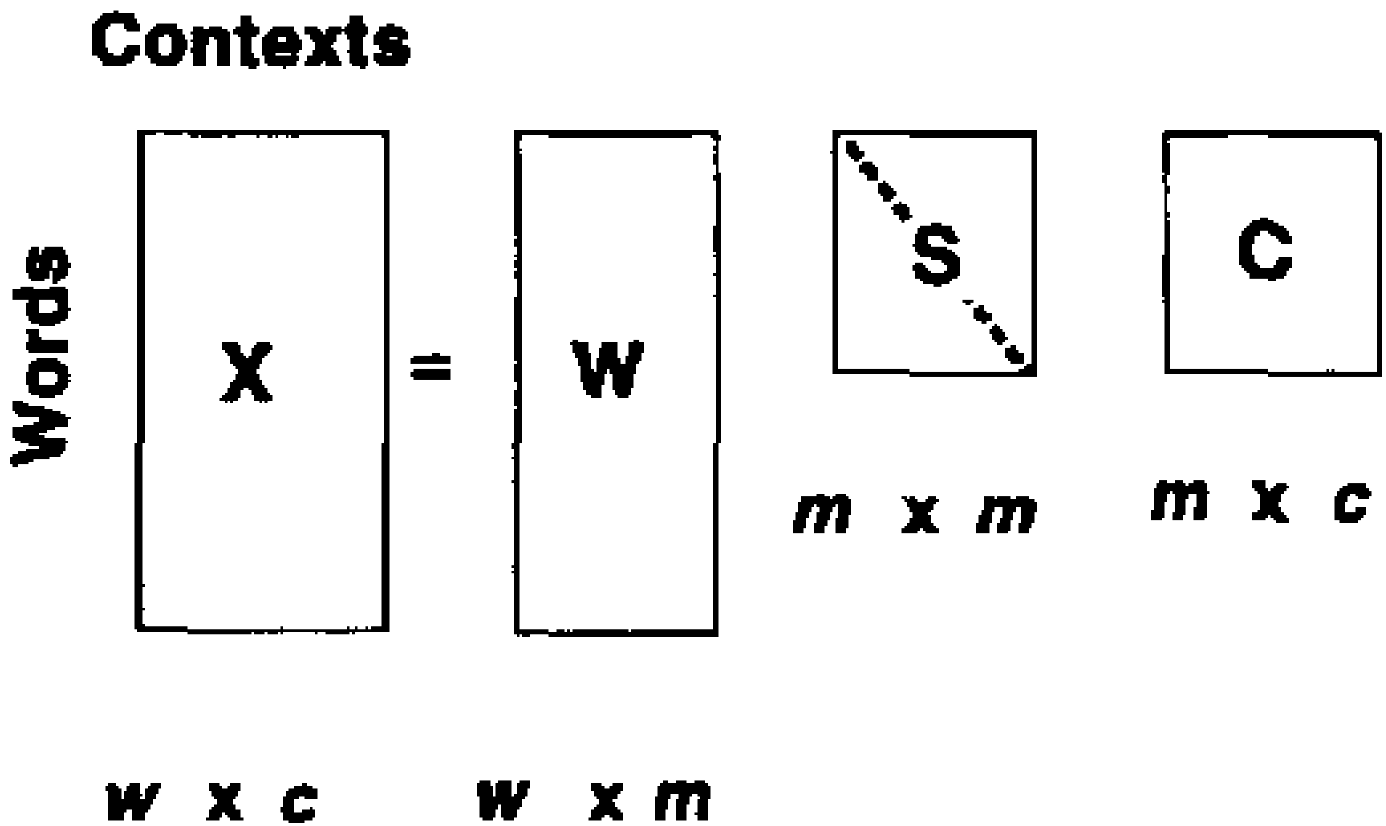
- Approximate an N-dimensional dataset using fewer dimensions
- By first rotating the axes into a new space
- In which the highest order dimension captures the most variance in the original dataset
- And the next dimension captures the next most variance, etc.
- Many such (related) methods:
  - PCA – principle components analysis
  - Factor Analysis
  - SVD

Dimensionality reduction



Singular Value Decomposition

Any $(w \times c)$ matrix X equals the product of 3 matrices:



Singular Value Decomposition

Any $(w \times c)$ matrix X equals the product of 3 matrices:

$$X = W\,S\,C$$

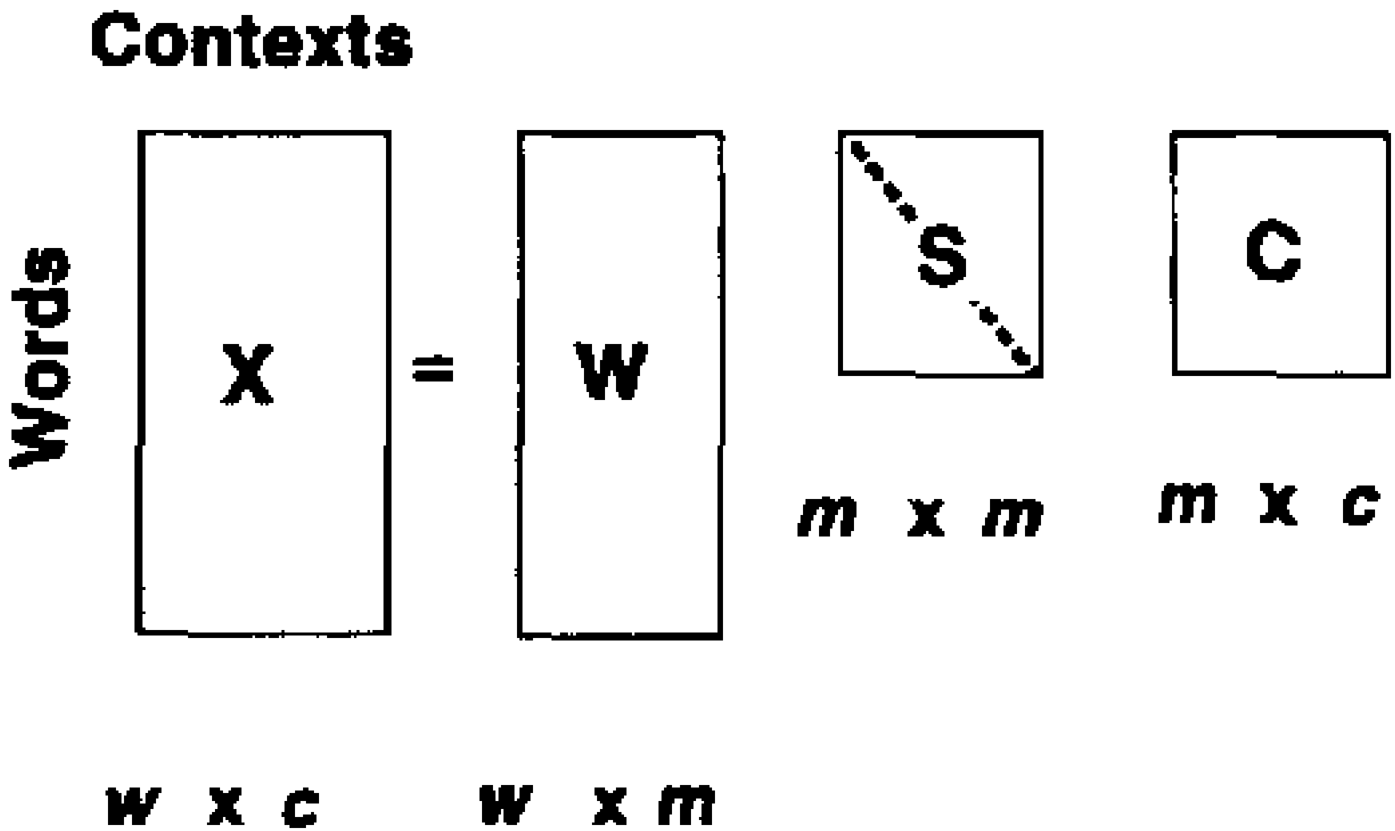
W: $(w \times m)$ matrix: rows corresponding to original but m columns represents a dimension in a new latent space, such that

- m column vectors are orthogonal to each other
- m = "Rank" of X.

S: $(m \times m)$ matrix: diagonal matrix of **singular values** expressing the importance of each dimension.

C: $(m \times c)$ matrix: columns corresponding to original but m rows corresponding to singular values

Singular Value Decomposition

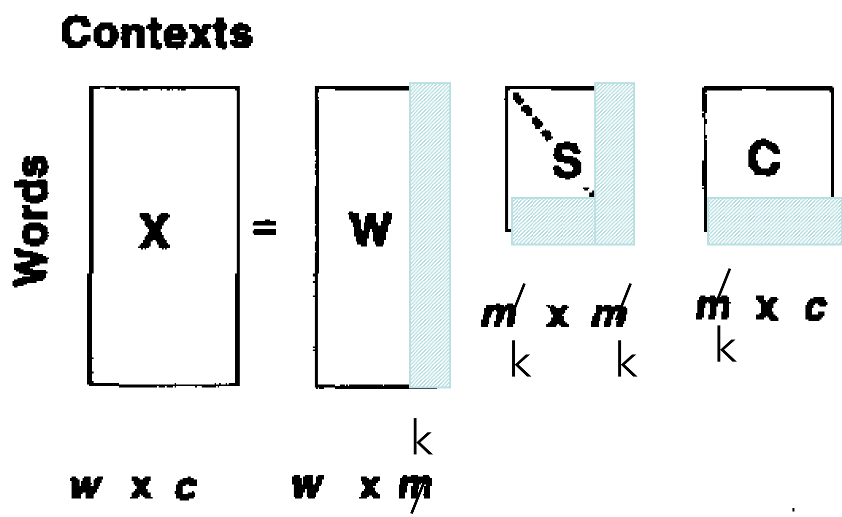


Landuaer and Dumais 1997

SVD applied to term-document matrix: Latent Semantic Analysis (LSA)

Deerwester et al (1988)

- Often m is not small enough!
- If instead of keeping all m dimensions, we just keep the top k singular values. Let's say 300.
- The result is a least-squares approximation to the original X
- But instead of multiplying, we'll just make use of W.
- Each row of W:
  - A k-dimensional vector
  - Representing word W



LSA more details

- 300 dimensions are commonly used
- The cells are commonly weighted by a product of two weights
  - Local weight: Log term frequency
  - Global weight: either idf or an entropy measure

Let’s return to PPMI word-word matrices

- Can we apply SVD to them?

SVD applied to term-term matrix

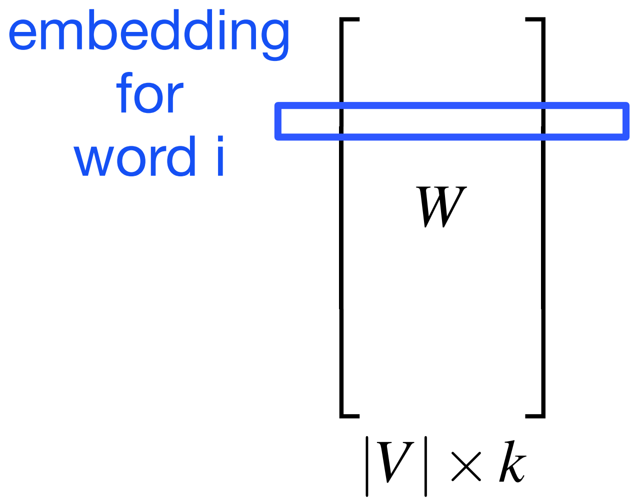
$$\underbrace{\begin{bmatrix} \\ \\ X \\ \\ \\ \end{bmatrix}}_{|V| \times |V|} = \underbrace{\begin{bmatrix} \\ \\ W \\ \\ \\ \end{bmatrix}}_{|V| \times |V|} \underbrace{\begin{bmatrix} \sigma_1 & 0 & 0 & \dots & 0 \\ 0 & \sigma_2 & 0 & \dots & 0 \\ 0 & 0 & \sigma_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \sigma_V \end{bmatrix}}_{|V| \times |V|} \underbrace{\begin{bmatrix} \\ \\ C \\ \\ \\ \end{bmatrix}}_{|V| \times |V|}$$

(simplifying assumption: the matrix has rank |V|)

Truncated SVD on term-term matrix

$$\underset{|V| \times |V|}{\begin{bmatrix} \\ \\ X \\ \\ \\ \end{bmatrix}} = \underset{|V| \times k}{\begin{bmatrix} \\ \\ W \\ \\ \\ \end{bmatrix}} \underset{k \times k}{\begin{bmatrix} \sigma_1 & 0 & 0 & \dots & 0 \\ 0 & \sigma_2 & 0 & \dots & 0 \\ 0 & 0 & \sigma_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \sigma_k \end{bmatrix}} \underset{k \times |V|}{\begin{bmatrix} & C & \end{bmatrix}}$$

Truncated SVD produces embeddings



- Each row of W matrix is a k-dimensional representation of each word *w*
- K might range from 50 to 1000
- Generally we keep the top k dimensions, but some experiments suggest that getting rid of the top 1 dimension or even the top 50 dimensions is helpful (Lapesa and Evert 2014).

Embeddings versus sparse vectors

Dense SVD embeddings sometimes work better than sparse PPMI matrices at tasks like word similarity

- Denoising: low-order dimensions may represent unimportant information
- Truncation may help the models generalize better to unseen data.
- Having a smaller number of dimensions may make it easier for classifiers to properly weight the dimensions for the task.
- Dense models may do better at capturing higher order co-occurrence.