Lecture 3

Terms Weighting -- TFIDF

Terms?

term	doc. freq.	→	postings lists
ambitious	1	→	2
be	1	→	2
brutus	2	→	1 → 2
capitol	1	→	1
caesar	2	→	1 → 2
did	1	→	1
enact	1	→	1
hath	1	→	2
i	1	→	1
i'	1	→	1
it	1	→	2
julius	1	→	1
killed	1	→	1
let	1	→	2
me	1	→	1
noble	1	→	2
so	1	→	2
the	2	→	1 → 2
told	1	→	2
you	1	→	2
was	2	→	1 → 2
with	1	→	2

Lists of docIDs

Terms and counts

Pointers

Zipf Distribution

- The Important Points:
  - a few elements occur *very frequently*
  - a medium number of elements have medium frequency
  - many elements occur *very infrequently*

Observation: MANY phenomena can be characterized this way.

- Words in a text collection
- Library book checkout patterns
- Incoming Web Page Requests (Nielsen)
- Outgoing Web Page Requests (Cunha & Crovella)
- Document Size on Web (Cunha & Crovella)

Sample Word Frequency Data

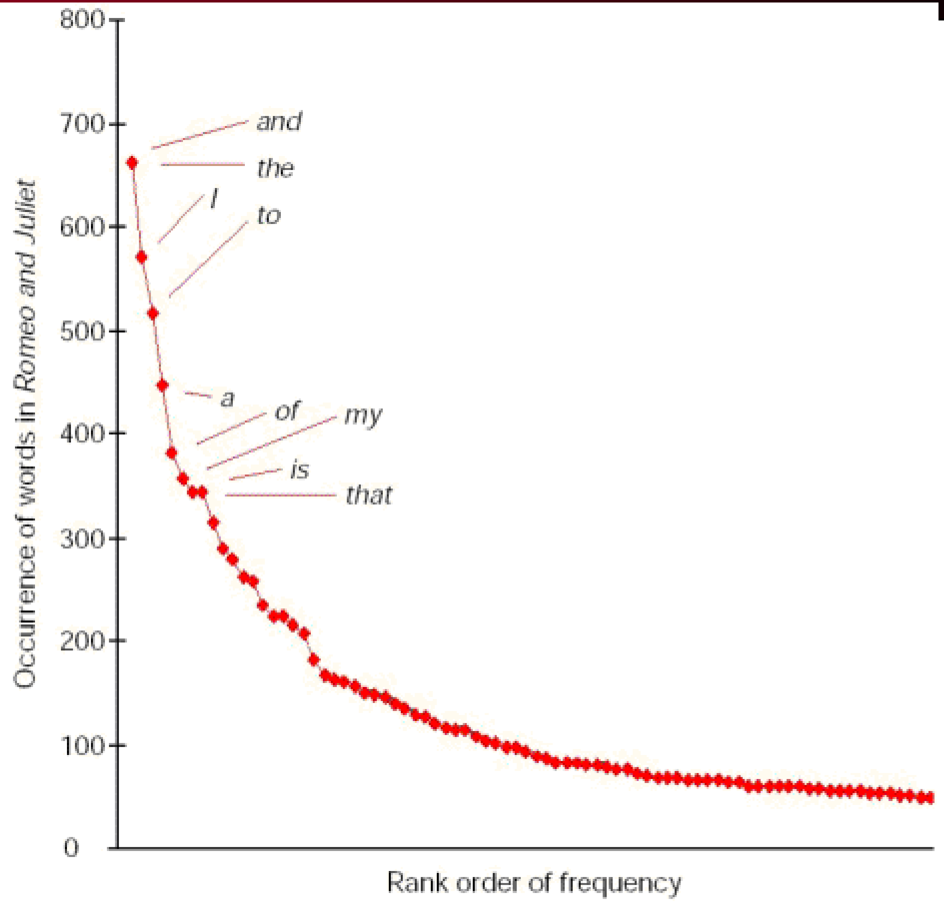
(from B. Croft, UMass)

Frequent Word	Number of Occurrences	Percentage of Total
the	7,398,934	5.9
of	3,893,790	3.1
to	3,364,653	2.7
and	3,320,687	2.6
in	2,311,785	1.8
is	1,559,147	1.2
for	1,313,561	1.0
The	1,144,860	0.9
that	1,066,503	0.8
said	1,027,713	0.8

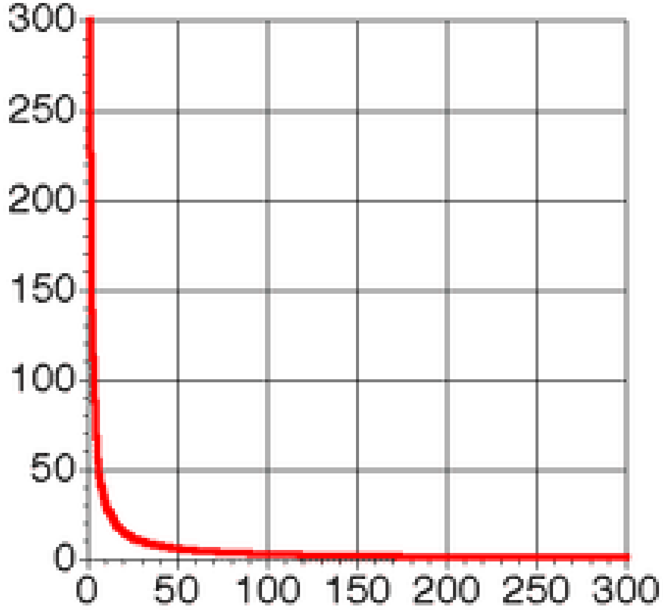
Frequencies from 336,310 documents in the 1GB TREC Volume 3 Corpus
125,720,891 total word occurrences; 508,209 unique words

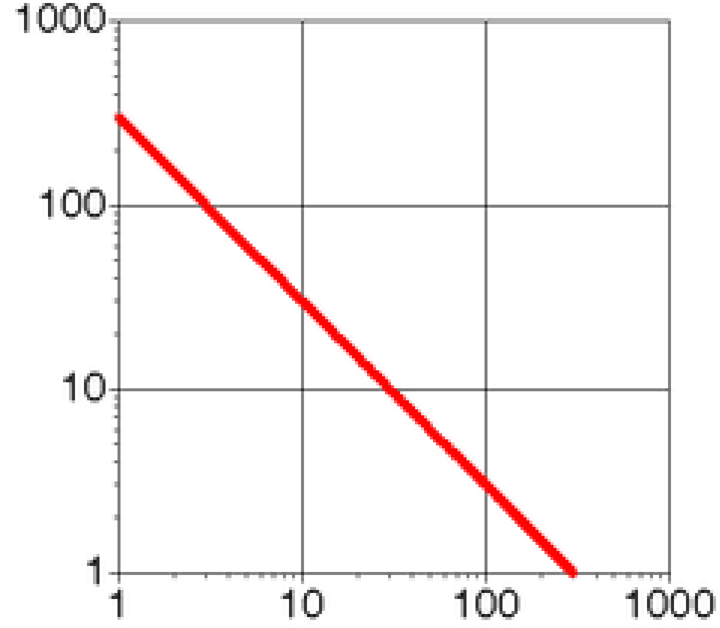
Zipf Distribution

The product of the frequency of words (f) and their rank (r) is approximately constant



Zipf Distribution (linear and log scale)





Zipf Distribution

- The product of the frequency of words (f) and their rank (r) is approximately constant
  - Rank = order of words' frequency of occurrence

$$f = C * 1/r$$

$$C \cong N/10$$

- Another way to state this is with an approximately correct rule of thumb:
  - Say the most common term occurs C times
  - The second most common occurs C/2 times
  - The third most common occurs C/3 times
  - …

Very frequent word stems

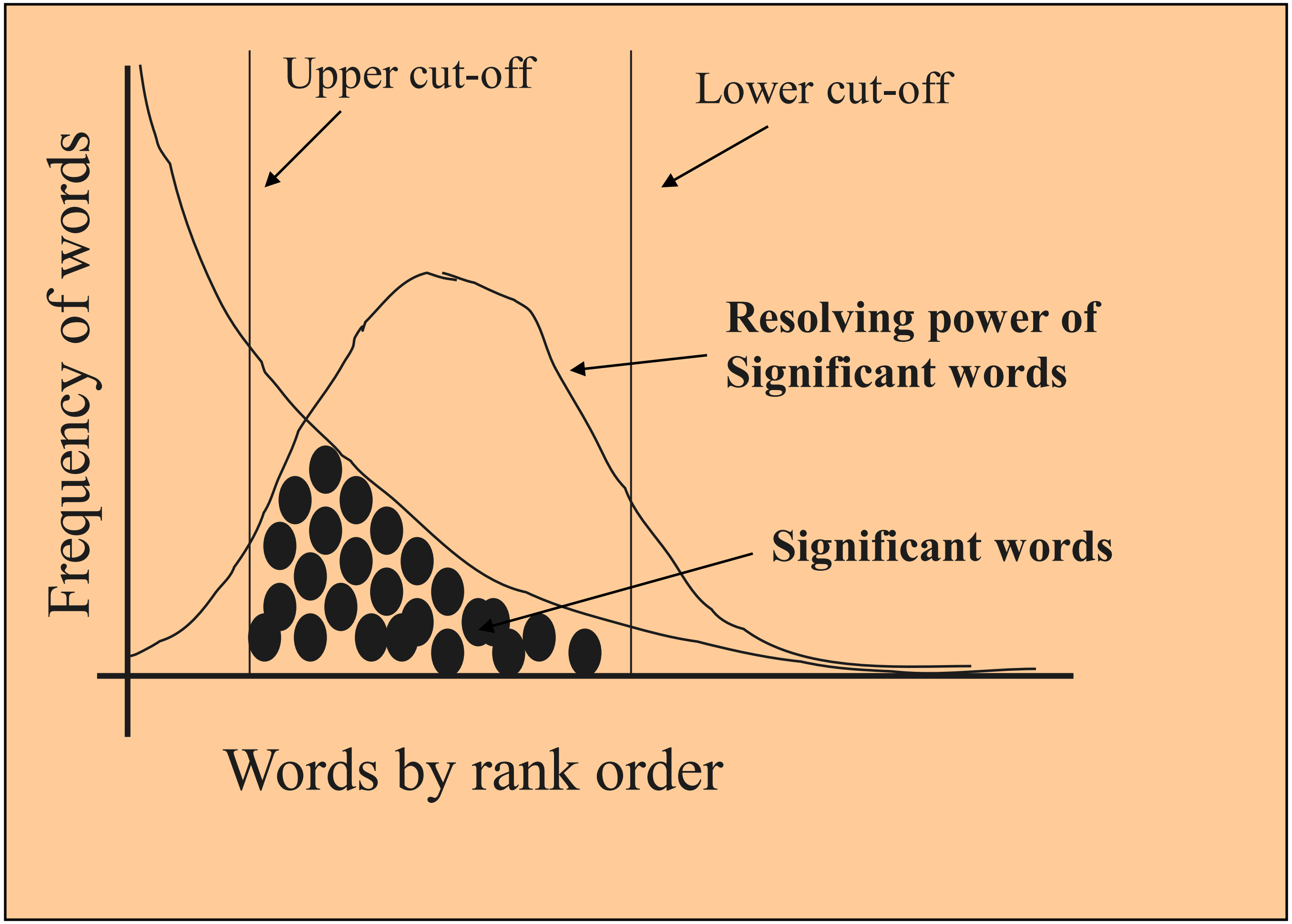
WORD	FREQ
u	63245
ha	65470
california	67251
m	67903
1998	68662
system	69345
t	70014
about	70923
servic	71822
work	71958
home	72131
other	72726
research	74264
1997	75323
can	76762
next	77973
your	78489
all	79993
public	81427
us	82551
c	83250
www	87029
wa	92384
program	95260
not	100204
http	100696
d	101034
html	103698
student	104635
univers	105183
inform	106463
will	109700
new	115937
have	119428
page	128702
messag	141542
from	147440
you	162499
edu	167298
be	185162
publib	189334
librari	189347
i	190635
lib	223851
that	227311
s	234467
berkelei	245406
re	272123
web	280966
archiv	305834

Words that occur few times

WORD	FREQ
agenda□augu	1
an□electronic	1
center□□janu	1
packard□equi	1
system□□july	1
systems□cs1	1
today□mcb	1
workshops□fi	1
workshops□th	1
□lollini	1
0+	1
0	1
00summary	1
35816	1
35823	1
01d	1
35830	1
35837	1
02-156-10	1
35844	1
35851	1
02aframst	1
311	1
313	1
03agenvchm	1
401	1
408	1
408	1
422	1
424	1
429	1
04agrcecon	1
04cklist	1
05-128-10	1
501	1
506	1
05amstud	1
06anhist	1
07-149	1
07-800-80	1
07anthro	1
08apst	1

Word Frequency vs. Resolving Power
(from van Rijsbergen 79)

The most frequent words are *not* the most descriptive.



Statistical Independence

Two events x and y are statistically independent if the product of their probability of their happening individually equals their probability of happening together.

$$\boxed{P(x)P(y) = P(x, y)}$$

Indexing

- indexing: assign identifiers to text items.
- assign: manual vs. automatic indexing
- identifiers:
  - objective vs. nonobjective text identifiers
    cataloging rules define, e.g., author names, publisher names, dates of publications, …
  - controlled vs. uncontrolled vocabularies
    instruction manuals, terminological schedules, …
  - single-term vs. term phrase

Two Issues

- Issue 1: indexing exhaustivity
  - exhaustive: assign a large number of terms
  - nonexhaustive
- Issue 2: term specificity
  - broad terms (generic)
    cannot distinguish relevant from nonrelevant items
  - narrow terms (specific)
    retrieve relatively fewer items, but most of them are relevant

Parameters of retrieval effectiveness

- Recall

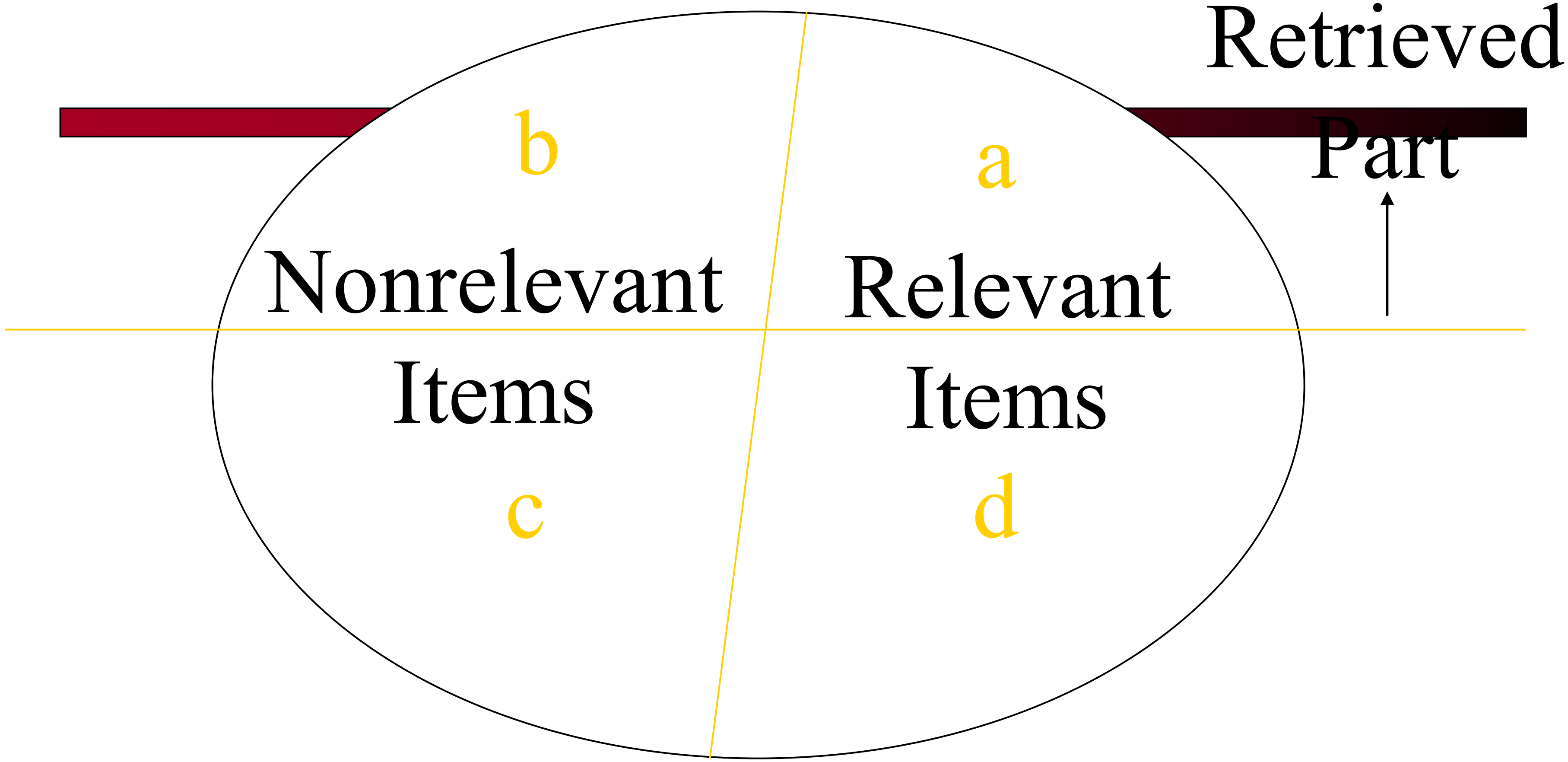
$$R = \frac{\text{Number of relevant items retrieved}}{\text{Total number of relevant items in collection}}$$

- Precision

$$P = \frac{\text{Number of relevant items retrieved}}{\text{Total number of items retrieved}}$$

- Goal

  high recall and high precision



$$\text{Recall} = \frac{a}{a+d} \qquad \text{Precision} = \frac{a}{a+b}$$

A Joint Measure

- F-score

$$F = \frac{(\beta^2 + 1) \times P \times R}{\beta^2 \times P + R}$$

  - β is a parameter that encode the importance of recall and procedure.
  - β=1: equal weight
  - β>1: precision is more important
  - β<1: recall is more important

Choices of Recall and Precision

- Both recall and precision vary from 0 to 1.
- In principle, the average user wants to achieve both high recall and high precision.
- In practice, a compromise must be reached because simultaneously optimizing recall and precision is not normally achievable.

Choices of Recall and Precision (*Continued*)

- Particular choices of indexing and search policies have produced variations in performance ranging from 0.8 precision and 0.2 recall to 0.1 precision and 0.8 recall.
- In many circumstance, both the recall and the precision varying between 0.5 and 0.6 are more satisfactory for the average users.

Term-Frequency Consideration

- Function words
  - for example, "and", "or", "of", "but", …
  - the frequencies of these words are high in all texts
- Content words
  - words that actually relate to document content
  - varying frequencies in the different texts of a collect
  - indicate term importance for content

A Frequency-Based Indexing Method

- Eliminate common function words from the document texts by consulting a special dictionary, or stop list, containing a list of high frequency function words.
- Compute the term frequency tf_{ij} for all remaining terms T_j in each document D_i, specifying the number of occurrences of T_j in D_i.
- Choose a threshold frequency T, and assign to each document D_i all term T_j for which $tf_{ij} > T$.

Discussions

- high-frequency terms
  favor recall
- high precision
  the ability to distinguish individual documents from each other
- high-frequency terms
  good for precision when its term frequency is not equally high in all documents.

Ranked retrieval

- Thus far, our queries have all been Boolean.
  - Documents either match or don’t.
- Good for expert users with precise understanding of their needs and the collection.
- Also good for applications: Applications can easily consume 1000s of results.
  - Not good for the majority of users.
  - Most users incapable of writing Boolean queries (or they are, but they think it’s too much work).
- Most users don’t want to wade through 1000s of results.
  - This is particularly true of web search.

Problem with Boolean search: feast or famine

- Boolean queries often result in either too few (=0) or too many (1000s) results.
- Query 1: "*standard user iPhone6* " → 200,000 hits
- Query 2: "*standard user iPhone6 no SIMcard found*": 0 hits
- It takes skill to come up with a query that produces a manageable number of hits.
- With a ranked list of documents it does not matter how large the retrieved set is.

Scoring as the basis of ranked retrieval

- We wish to return in order the documents most likely to be useful to the searcher
- How can we rank-order the documents in the collection with respect to a query?
- Assign a score – say in [0, 1] – to each document
- This score measures how well document and query “match”.

Query-document matching scores

- We need a way of assigning a score to a query/document pair
- Let's start with a one-term query
- If the query term does not occur in the document: score should be 0
- The more frequent the query term in the document, the higher the score (should be)
- We will look at a number of alternatives for this.

Jaccard coefficient

- Recall set similarity: A commonly used measure of overlap of two sets A and B
- jaccard$(A,B) = |A \cap B| / |A \cup B|$
- jaccard$(A,A) = 1$
- jaccard$(A,B) = 0$ if $A \cap B = 0$
- A and B don't have to be the same size.
- Always assigns a number between 0 and 1.

Jaccard coefficient: Scoring example

- What is the query-document match score that the Jaccard coefficient computes for each of the two documents below?
- Query: *ides of march*
- Document 1: *caesar died in march*
- Document 2: *the long march*

Issues with Jaccard for scoring

- It doesn't consider term frequency (how many times a term occurs in a document)
- Rare terms in a collection are more informative than frequent terms. Jaccard doesn't consider this information
- We need a more sophisticated way of normalizing for length
- Later in this lecture, we'll use $|A \cap B| / \sqrt{|A \cup B|}$
- . . . instead of $|A \cap B|/|A \cup B|$ (Jaccard) for length normalization.

Recall (Lecture 1): Binary term-document incidence matrix

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

Each document is represented by a binary vector $\in \{0,1\}^{|V|}$

Term-document count matrices

- Consider the number of occurrences of a term in a document:
  - Each document is a $\boxed{\text{count vector}}$ in $\mathbb{N}^{v}$: a column below

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	157	73	0	0	0	0
Brutus	4	157	0	1	0	0
Caesar	232	227	0	2	1	1
Calpurnia	0	10	0	0	0	0
Cleopatra	57	0	0	0	0	0
mercy	2	0	3	5	5	1
worser	2	0	1	1	1	0

Term-document count matrices

- Consider the number of occurrences of a term in a document:
  - Each document is a count vector in $\mathbb{N}^{V}$: a column below

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	157	73	0	0	0	0
Brutus	4	157	0	1	0	0
Caesar	232	227	0	2	1	1
Calpurnia	0	10	0	0	0	0
Cleopatra	57	0	0	0	0	0
mercy	2	0	3	5	5	1
worser	2	0	1	1	1	0

Bag of words model

- Vector representation doesn't consider the ordering of words in a document
- *John is quicker than Mary* and *Mary is quicker than John* have the same vectors
- This is called the bag of words model.
- In a sense, this is a step back: The positional index was able to distinguish these two documents.
- We will look at "recovering" positional information later in this course.
- For now: bag of words model

Term frequency tf

- The term frequency $tf_{t,d}$ of term *t* in document *d* is defined as the number of times that *t* occurs in *d*.
- We want to use tf when computing query-document match scores. But how?
- Raw term frequency is not what we want:
  - A document with 10 occurrences of the term is more relevant than a document with 1 occurrence of the term.
  - But not 10 times more relevant.
- Relevance does not increase proportionally with term frequency.

NB: frequency = count in IR

Log-frequency weighting

- The log frequency weight of term t in d is

$$w_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{tf}_{t,d}, & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0, & \text{otherwise} \end{cases}$$

- $0 \rightarrow 0$, $1 \rightarrow 1$, $2 \rightarrow 1.3$, $10 \rightarrow 2$, $1000 \rightarrow 4$, etc.
- Score for a document-query pair: sum over terms *t* in both *q* and *d*:
- score $= \sum_{t \in q \cap d} (1 + \log \mathrm{tf}_{t,d})$
- The score is 0 if none of the query terms is present in the document.

Document frequency

- Rare terms are more informative than frequent terms
  - Recall stop words
- Consider a term in the query that is rare in the collection (e.g., *arachnocentric*)
- A document containing this term is very likely to be relevant to the query *arachnocentric*
- → We want a high weight for rare terms like *arachnocentric*.

Document frequency, continued

- Consider a query term that is frequent in the collection (e.g., *high, increase, line*)
- A document containing such a term is more likely to be relevant than a document that doesn't, but it's not a sure indicator of relevance.
- → For frequent terms, we want positive weights for words like *high, increase, and line*, but lower weights than for rare terms.
- We will use document frequency (df) to capture this in the score.
- df ($\leq N$) is the number of documents that contain the term

idf weight

- df_t is the document frequency of t: the number of documents that contain t
  - df is a measure of the informativeness of t
- We define the idf (inverse document frequency) of t by

$$\mathrm{idf}_t = \log_{10} N/\mathrm{df}_t$$

  - We use log N/df_t instead of N/df_t to "dampen" the effect of idf.

Will turn out the base of the log is immaterial.

idf example, suppose N= 1 million

term	df_t	idf_t
calpurnia	1	6
animal	100	4
sunday	1,000	3
fly	10,000	2
under	100,000	1
the	1,000,000	0

There is one idf value for each term *t* in a collection.

Collection vs. Document frequency

- The collection frequency of *t* is the number of occurrences of *t* in the collection, counting multiple occurrences.
- Example:

Word	Collection frequency	Document frequency
insurance	10440	3997
try	10422	8760

- Which word is a better search term (and should get a higher weight)?

tf-idf weighting

- The tf-idf weight of a term is the product of its tf weight and its idf weight.

$$\mathrm{w}_{t,d} = (1 + \log \mathrm{tf}_{t,d}) \times \log_{10} N / \mathrm{df}_t$$

- Best known weighting scheme in information retrieval
- Note: the "-" in tf-idf is a hyphen, not a minus sign!
- Alternative names: tf.idf, tf x idf
- Increases with the number of occurrences within a document
- Increases with the rarity of the term in the collection

Binary → count → weight matrix

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	5.25	3.18	0	0	0	0.35
Brutus	1.21	6.1	0	1	0	0
Caesar	8.59	2.54	0	1.51	0.25	0
Calpurnia	0	1.54	0	0	0	0
Cleopatra	2.85	0	0	0	0	0
mercy	1.51	0	1.9	0.12	5.25	0.88
worser	1.37	0	0.11	4.15	0.25	1.95

Each document is now represented by a real-valued vector of tf-idf weights $\in \mathbb{R}^{|V|}$

Documents as vectors

- So we have a |V|-dimensional vector space
- Terms are axes of the space
- Documents are points or vectors in this space
- Very high-dimensional: hundreds of millions of dimensions when you apply this to a web search engine
- This is a very sparse vector - most entries are zero.

Queries as vectors

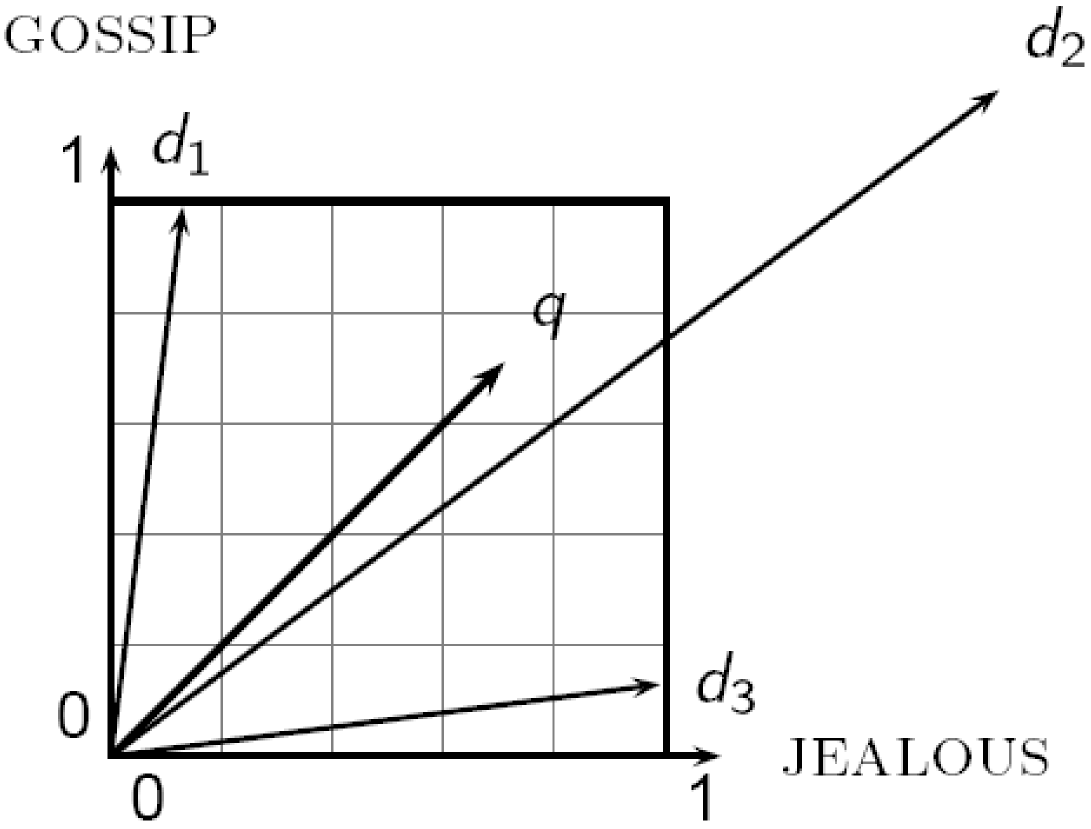
- Key idea 1: Do the same for queries: represent them as vectors in the space
- Key idea 2: Rank documents according to their proximity to the query in this space
- proximity = similarity of vectors
- proximity ≈ inverse of distance
- Recall: We do this because we want to get away from the you're-either-in-or-out Boolean model.
- Instead: rank more relevant documents higher than less relevant documents

Formalizing vector space proximity

- First cut: distance between two points
  - (= distance between the end points of the two vectors)
- Euclidean distance?
- Euclidean distance is a bad idea . . .
- . . . because Euclidean distance is large for vectors of different lengths.

Why distance is a bad idea

The Euclidean distance between $\vec{q}$ and $\vec{d_2}$ is large even though the distribution of terms in the query $\vec{q}$ and the distribution of terms in the document $\vec{d_2}$ are very similar.



Use angle instead of distance

- Thought experiment: take a document d and append it to itself. Call this document d′.
- "Semantically" d and d′ have the same content
- The Euclidean distance between the two documents can be quite large
- The angle between the two documents is 0, corresponding to maximal similarity.

- Key idea: Rank documents according to angle with query.

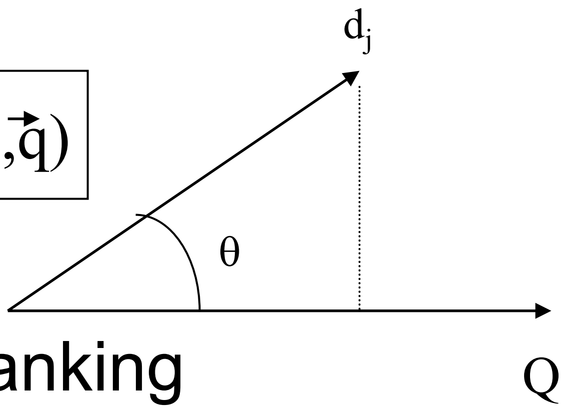
Similarity of document d_j w.r.t. query q

- The correlation between vectors $\vec{d_j}$ and $\vec{q}$

$$sim(d_j, q) = \frac{\vec{d}_j \bullet \vec{q}}{|\vec{d}_j| \times |\vec{q}|}$$

$$= \frac{\sum_{i=1}^{t} w_{i,j} \times w_{i,q}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2} \times \sqrt{\sum_{j=1}^{t} w_{i,q}^2}}$$

$$\boxed{\cos(\vec{d_j}, \vec{q})}$$



- $|\vec{q}|$ does not affect the ranking
- $|\vec{d_j}|$ provides a normalization

From angles to cosines

- The following two notions are equivalent.
  - Rank documents in <u>decreasing</u> order of the angle between query and document
  - Rank documents in <u>increasing</u> order of cosine(query,document)
- Cosine is a monotonically decreasing function for the interval [0^o, 180^o]

Length normalization

- A vector can be (length-) normalized by dividing each of its components by its length – for this we use the L_2 norm: $\|\vec{x}\|_2 = \sqrt{\sum_i x_i^2}$
- Dividing a vector by its L_2 norm makes it a unit (length) vector
- Effect on the two documents d and d′ (d appended to itself) from earlier slide: they have identical vectors after length-normalization.

cosine(query,document)

Dot product

Unit vectors

$$\cos(\vec{q},\vec{d}) = \frac{\vec{q}\bullet\vec{d}}{|\vec{q}||\vec{d}|} = \frac{\vec{q}}{|\vec{q}|}\bullet\frac{\vec{d}}{|\vec{d}|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2}\sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

q_i is the tf-idf weight of term *i* in the query
d_i is the tf-idf weight of term *i* in the document
$\cos(\vec{q},\vec{d})$ is the cosine similarity of $\vec{q}$ and $\vec{d}$ … or, equivalently, the cosine of the angle between $\vec{q}$ and $\vec{d}$.

Cosine similarity amongst 3 documents

How similar are the novels
SaS: *Sense and Sensibility*
PaP: *Pride and Prejudice*, and
WH: *Wuthering Heights*?

term	SaS	PaP	WH
affection	115	58	20
jealous	10	7	11
gossip	2	0	6
wuthering	0	0	38

Term frequencies (counts)

3 documents example contd.

Log frequency weighting

term	SaS	PaP	WH
affection	3.06	2.76	2.30
jealous	2.00	1.85	2.04
gossip	1.30	0	1.78
wuthering	0	0	2.58

After normalization

term	SaS	PaP	WH
affection	0.789	0.832	0.524
jealous	0.515	0.555	0.465
gossip	0.335	0	0.405
wuthering	0	0	0.588

cos(SaS,PaP) $\approx$
$0.789 * 0.832 + 0.515 * 0.555 + 0.335 * 0.0 + 0.0 * 0.0$
≈ 0.94

cos(SaS,WH) ≈ 0.79

cos(PaP,WH) ≈ 0.69

Why do we have cos(SaS,PaP) > cos(SAS,WH)?

Computing cosine scores

COSINESCORE(q)

float $Scores[N] = 0$
float $Length[N]$
for each query term t
do calculate $w_{t,q}$ and fetch postings list for t
  for each pair$(d, \mathrm{tf}_{t,d})$ in postings list
  do $Scores[d] += w_{t,d} \times w_{t,q}$
Read the array *Length*
for each d
do $Scores[d] = Scores[d]/Length[d]$
return Top K components of $Scores[]$

tf-idf weighting has many variants

Term frequency		Document frequency		Normalization	
n (natural)	$\mathrm{tf}_{t,d}$	n (no)	1	n (none)	1
l (logarithm)	$1 + \log(\mathrm{tf}_{t,d})$	t (idf)	$\log \frac{N}{\mathrm{df}_t}$	c (cosine)	$\frac{1}{\sqrt{w_1^2+w_2^2+\ldots+w_M^2}}$
a (augmented)	$0.5 + \frac{0.5 \times \mathrm{tf}_{t,d}}{\max_t(\mathrm{tf}_{t,d})}$	p (prob idf)	$\max\{0, \log \frac{N-\mathrm{df}_t}{\mathrm{df}_t}\}$	u (pivoted unique)	$1/u$
b (boolean)	$\begin{cases} 1 & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$			b (byte size)	$1/CharLength^{\alpha}$, $\alpha < 1$
L (log ave)	$\frac{1+\log(\mathrm{tf}_{t,d})}{1+\log(\mathrm{ave}_{t\in d}(\mathrm{tf}_{t,d}))}$				

Columns headed 'n' are acronyms for weight schemes.

Why is the base of the log in idf immaterial?

Weighting may differ in queries vs documents

- Many search engines allow for different weightings for queries vs documents
- To denote the combination in use in an engine, we use the notation qqq.ddd with the acronyms from the previous table
- Example: ltn.ltc means:
- Query: logarithmic tf (l in leftmost column), idf (t in second column), no normalization …
- Document logarithmic tf, no idf and cosine normalization

Is this a bad idea?

tf-idf example: ltn.lnc

Document: *car insurance auto insurance*
Query: *best car insurance*

Term	Query					Document				Prod
	tf-raw	tf-wt	df	idf	wt	tf-raw	tf-wt	wt	n'lized	
auto	0	0	5000	2.3	0	1	1	1	0.52	0
best	1	1	50000	1.3	1.3	0	0	0	0	0
car	1	1	10000	2.0	2.0	1	1	1	0.52	1.04
insurance	1	1	1000	3.0	3.0	2	1.3	1	0.68	2.04

Exercise: what is *N*, the number of docs?

Doc length = $\sqrt{1^2+0^2+1^2+1^2} \approx 1.92$

Score = 0+0+1.04+2.04 = 3.08

Summary – vector space ranking

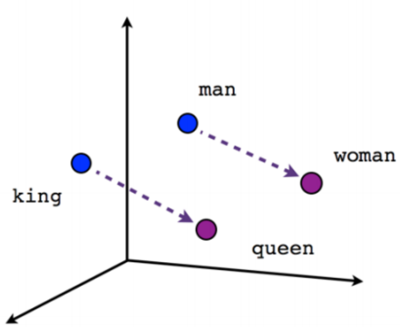
- Represent the query as a weighted tf-idf vector
- Represent each document as a weighted tf-idf vector
- Compute the cosine similarity score for the query vector and each document vector
- Rank documents with respect to the query by score
- Return the top K (e.g., $K = 10$) to the user

Vector Representation of Text

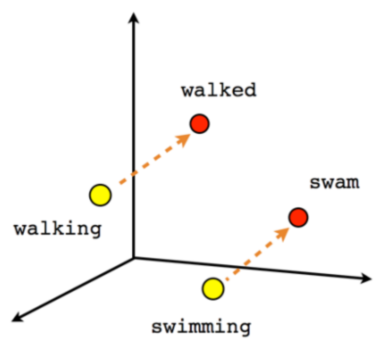
Word Embedding Technique
(word2vec)

Word to vector (word2vector)

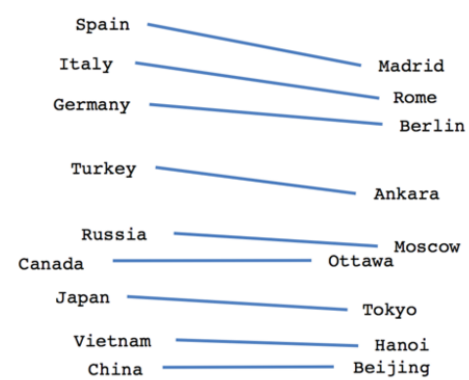
- The more often two words co-occur, the closer their vectors will be
- Two words have close meanings if their local neighborhoods are similar



Male-Female



Verb tense



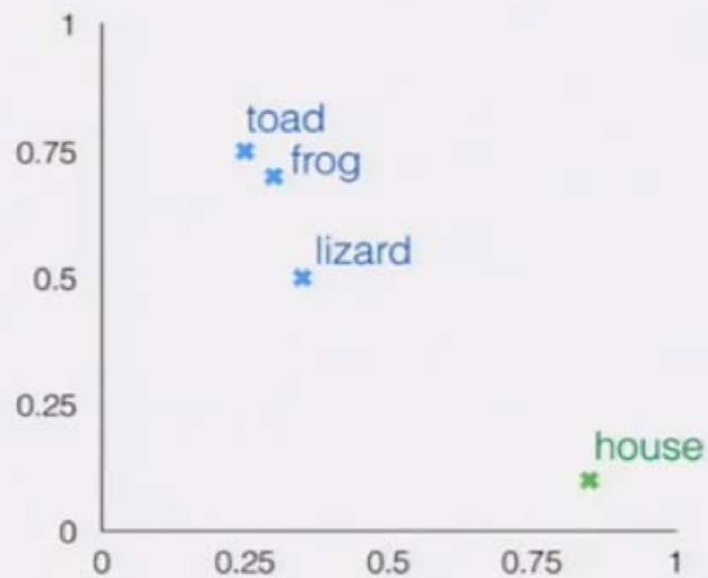
Country-Capital

Problem?

Distributed representations

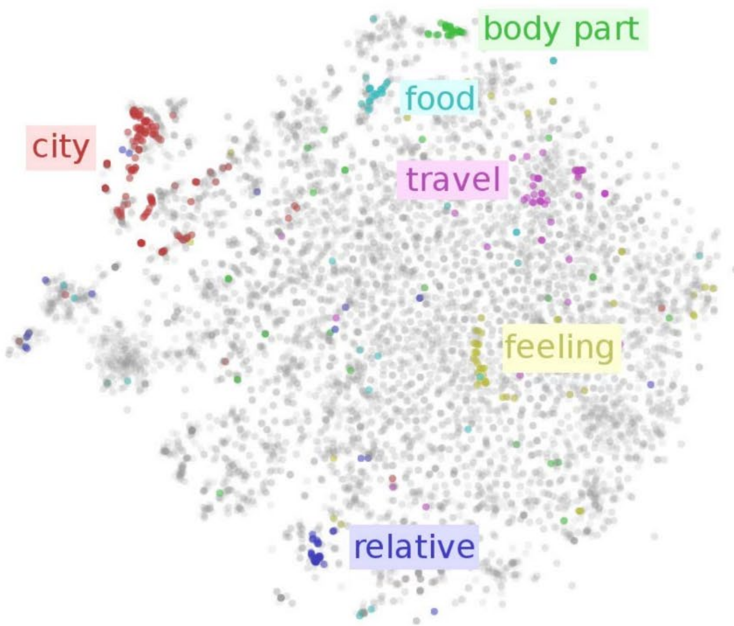
Word vectors aren't guaranteed to encode any linguistic relationships between words, but many models produce vectors that do

frog	[0.30 0.70]
toad	[0.25 0.75]
lizard	[0.35 0.50]
house	[0.85 0.10]



Example

Any technique mapping a word (or phrase) from it's original high-dimensional input space (the body of all words) to a lower-dimensional numerical vector space - so one *embeds* the word in a different space



Source: http://sebastianruder.com/content/images/2016/04/word_embeddings_colah.png

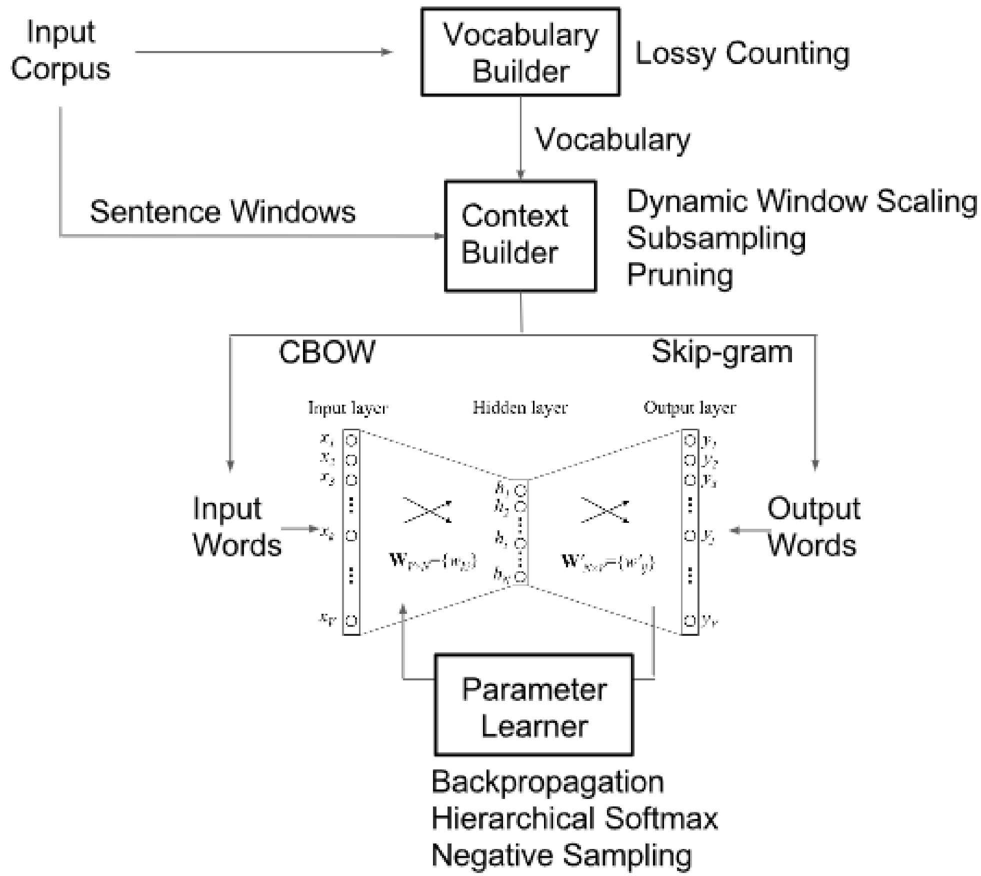
Points: original word space

Colored points / clusters: Word embedding

Word Representations

Traditional Method - Bag of Words Model	Word Embeddings
• Uses one hot encoding • Each word in the vocabulary is represented by one bit position in a HUGE vector. • For example, if we have a vocabulary of 10000 words, and "Hello" is the 4th word in the dictionary, it would be represented by: 0 0 0 1 0 0 . . . . 0 0 0 0 • Context information is not utilized	• Stores each word in as a point in space, where it is represented by a vector of fixed number of dimensions (generally 300) • Unsupervised, built just by reading huge corpus • For example, "Hello" might be represented as : [0.4, -0.11, 0.55, 0.3 . . . 0.1, 0.02] • Dimensions are basically projections along different axes, more of a mathematical concept.

Architecture



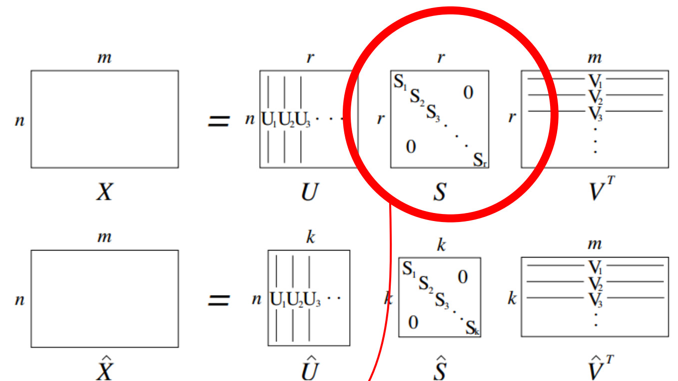
To compare pieces of text

- We need effective representation of
  - Words
  - Sentences
  - Text
- Approach 1: Use existing thesauri or ontologies like WordNet and Snomed CT (for medical).

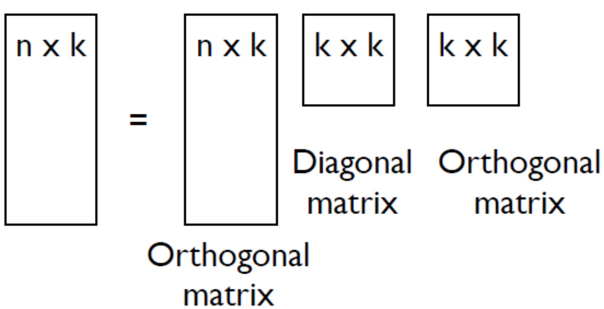
  Drawbacks:
  - Manual
  - Not context specific
- Approach 2: Use co-occurrences for word similarity. Drawbacks:
  - Quadratic space needed
  - Relative position and order of words not considered

Approach 3: low dimensional vectors

- Store only “important” information in fixed, low dimensional vector.
- Singular Value Decomposition (SVD) on co-occurrence matrix
  - $\hat{X}$ is the best rank k approximation to X , in terms of least squares
  - Motel = [0.286, 0.792, -0.177, -0.107, 0.109, -0.542, 0.349, 0.271]
- m = n = size of vocabulary
- $\hat{S}$ is the same matrix as S except that it contains only the top largest singular values

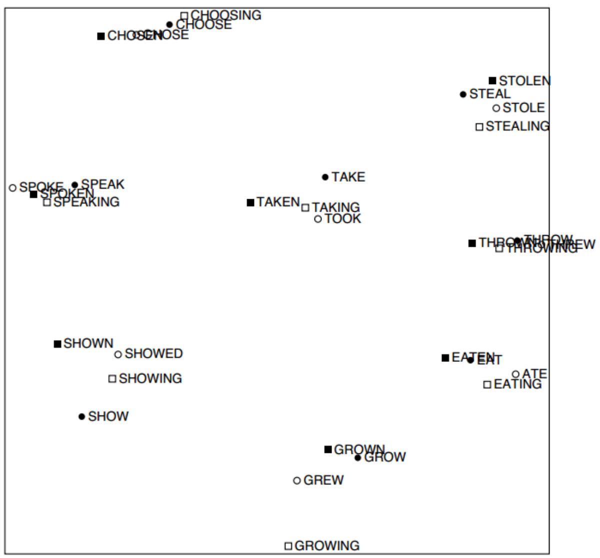


$$\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^\mathsf{T}$$



Example of Approach 3: low dimensional vectors

- An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence [Rohde et al. 2005]



Problems with SVD

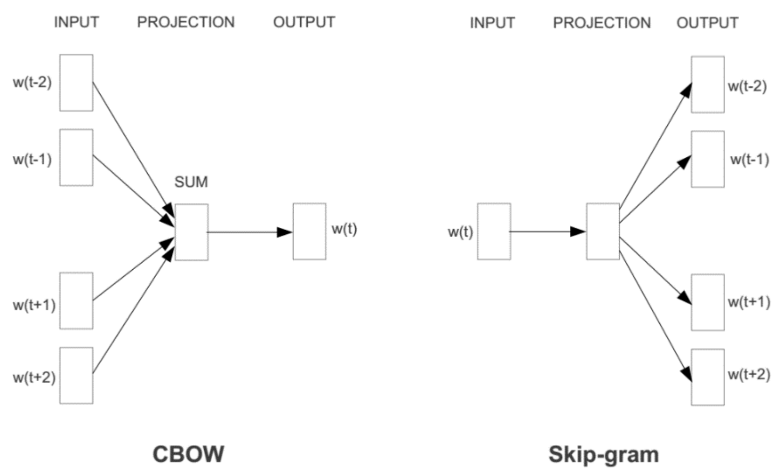
- Computational cost scales quadratically for n x m matrix: $O(mn^2)$ flops (when n<m)
- Hard to incorporate new words or documents
- Does not consider order of words

word2vec approach to represent the meaning of word

- Represent each word with a low-dimensional vector
- Word similarity = vector similarity
- Key idea: Predict surrounding words of every word
- Faster and can easily incorporate a new sentence/document or add a word to the vocabulary

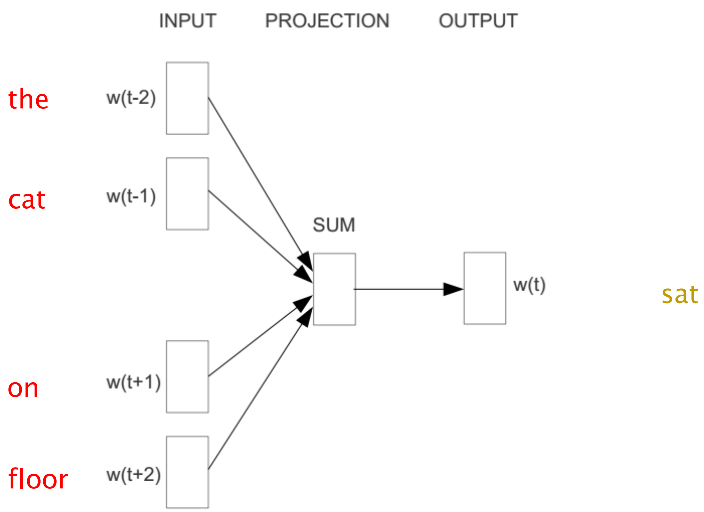
Represent the meaning of **word** – word2vec

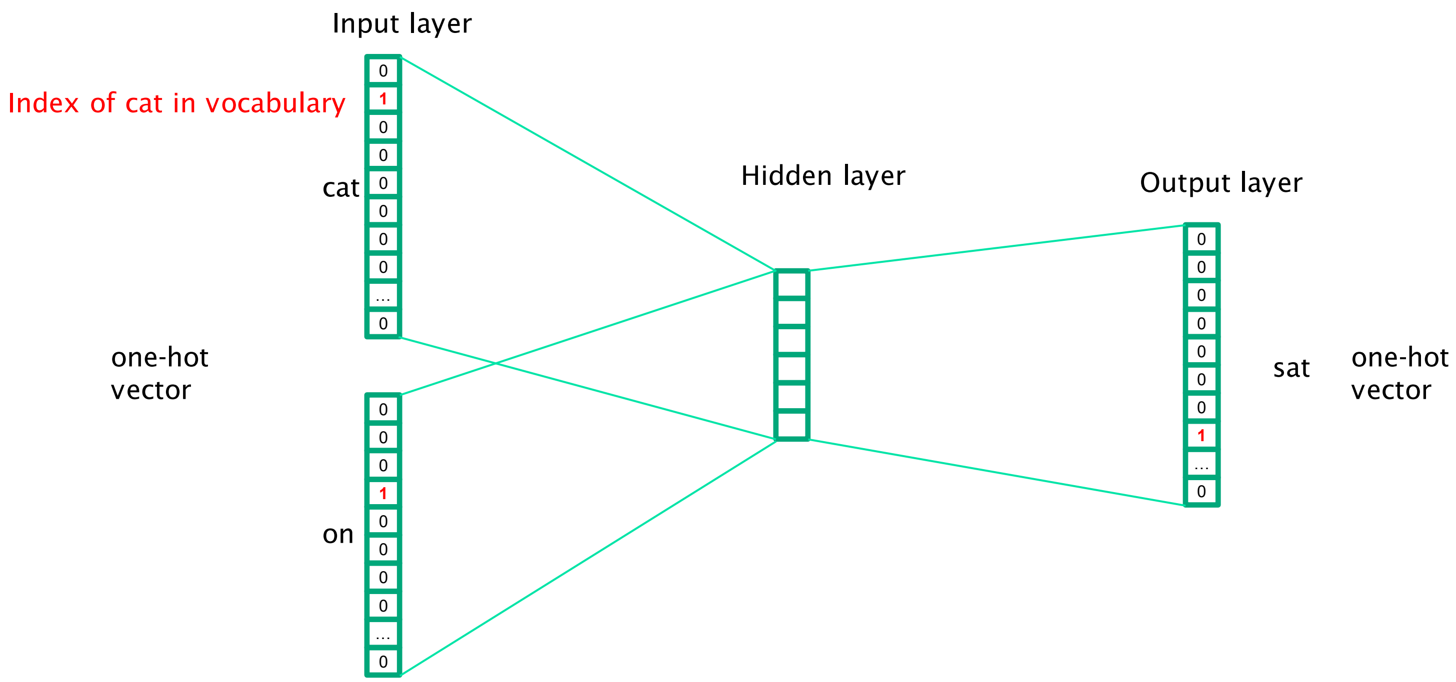
- 2 basic neural network models:
  - Continuous Bag of Word (CBOW): use a window of word to predict the middle word
  - Skip-gram (SG): use a word to predict the surrounding ones in window.

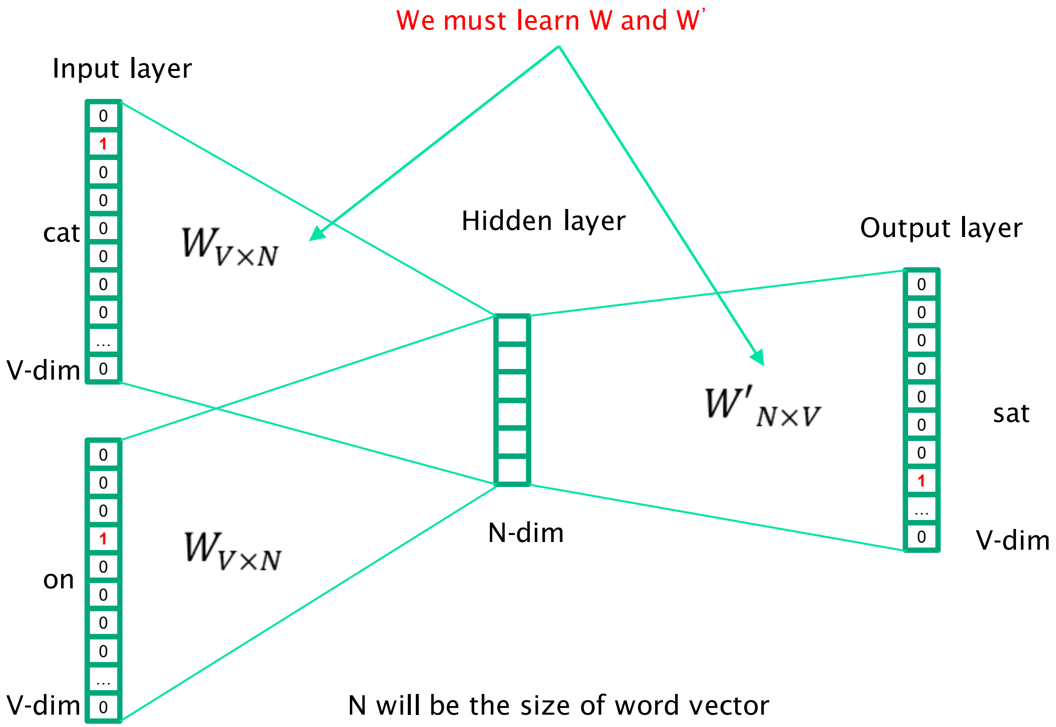


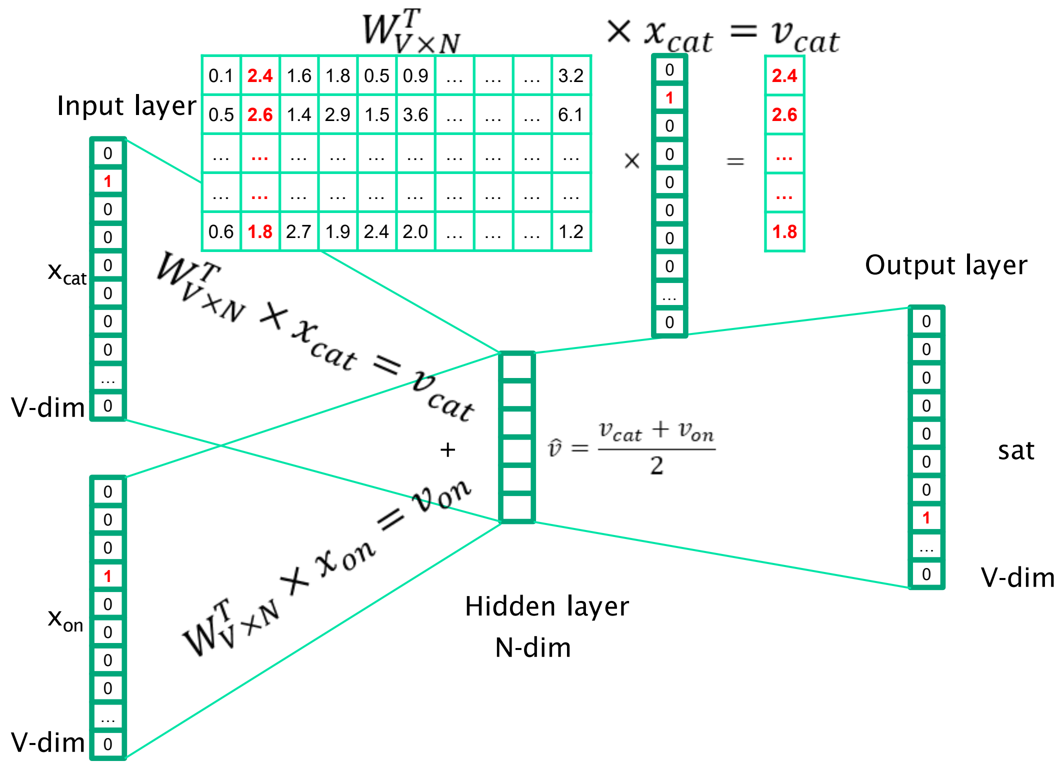
Word2vec – Continuous Bag of Word

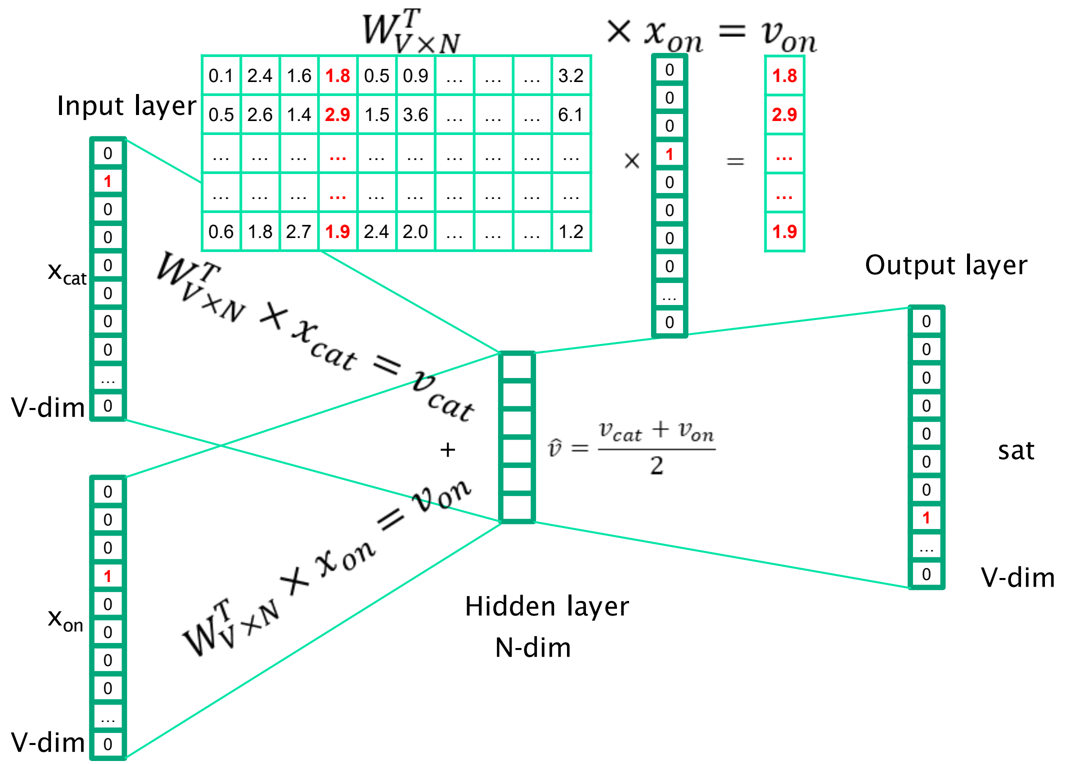
- E.g. "The cat sat on floor"
  - Window size = 2



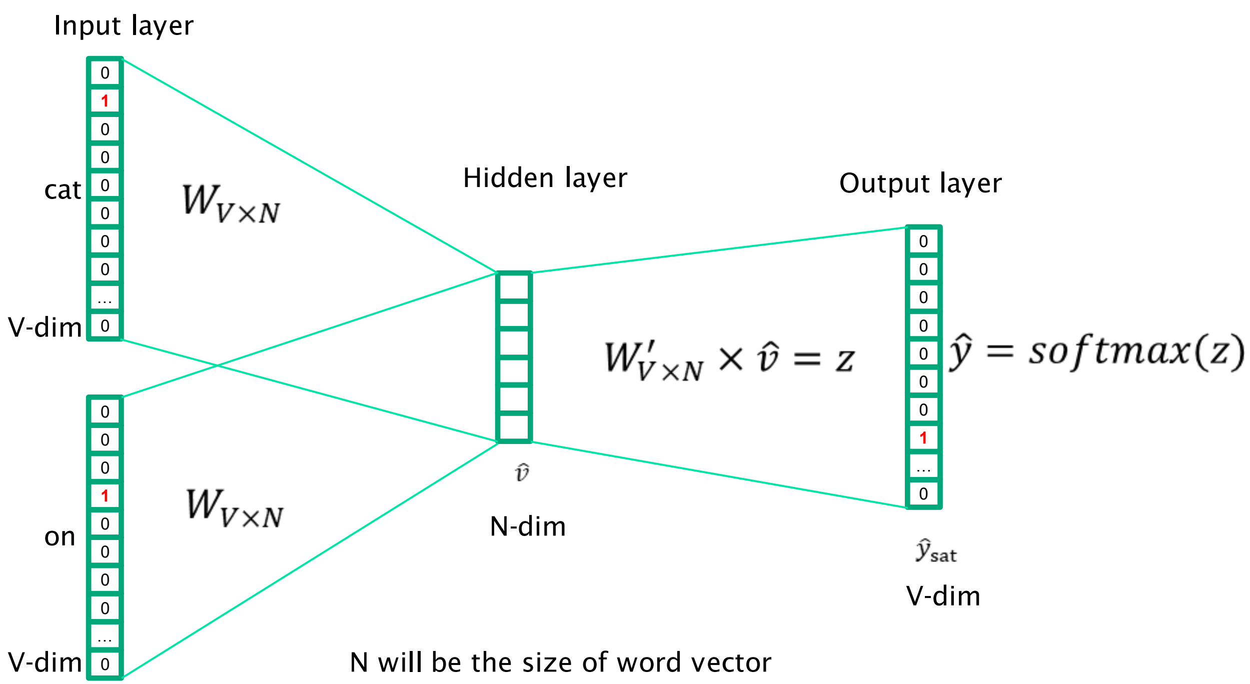

Input layer
Index of cat in vocabulary
cat
0
1
0
0
0
0
0
0
...
0
one-hot
vector
on
0
0
0
1
0
0
0
0
...
0
Hidden layer
Output layer
0
0
0
0
0
0
0
1
...
0
sat
one-hot
vector

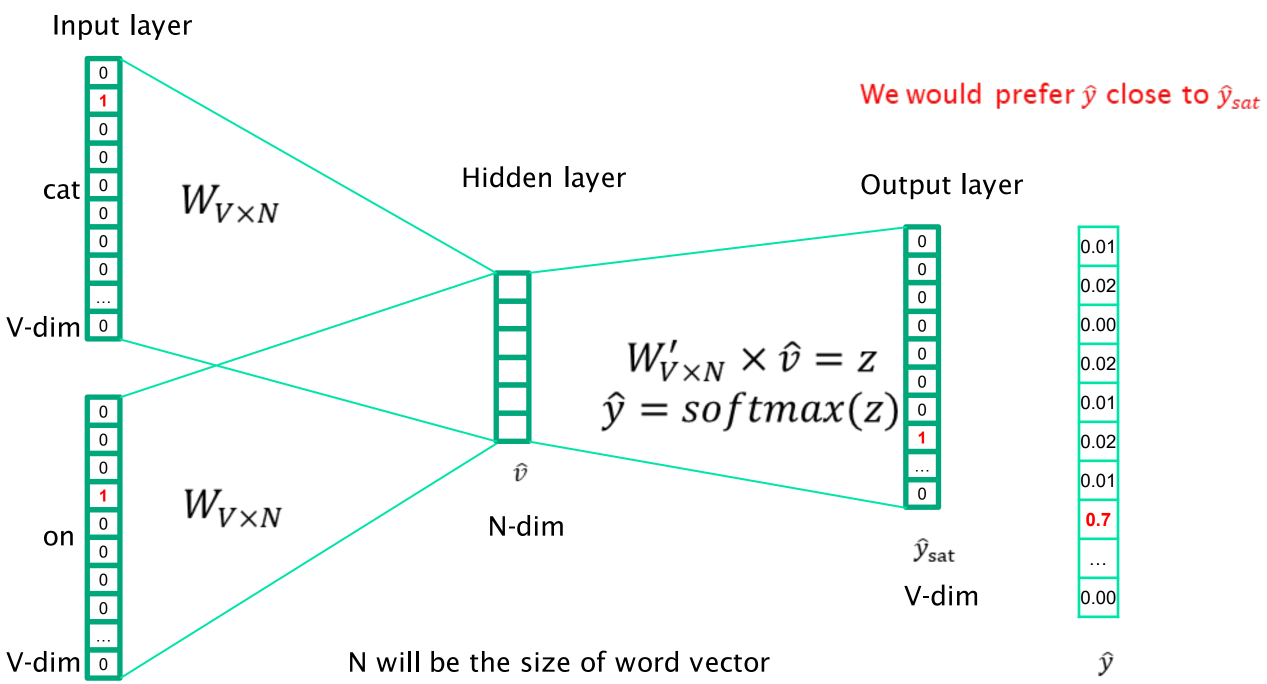

We must learn W and W'
Input layer
cat
V-dim
on
V-dim
$W_{V\times N}$
$W_{V\times N}$
Hidden layer
N-dim
Output layer
sat
V-dim
$W'_{N\times V}$
N will be the size of word vector

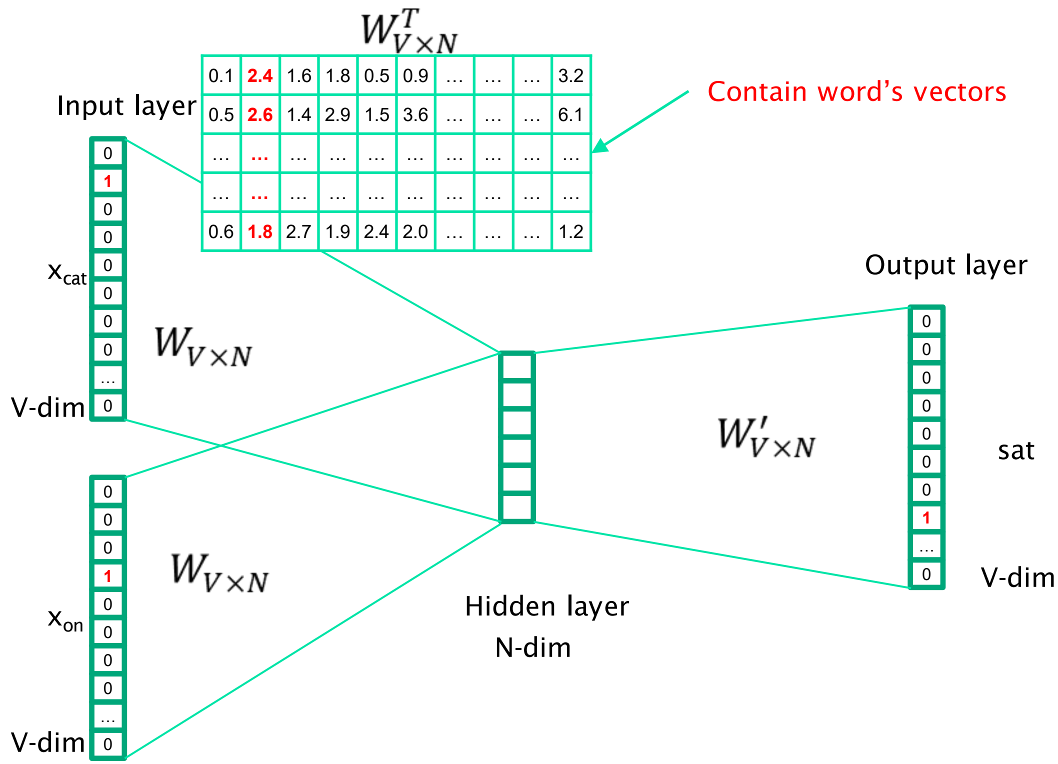

$W^T_{V\times N} \times x_{cat} = v_{cat}$
Input layer
0.1 2.4 1.6 1.8 0.5 0.9 … … … 3.2
0.5 2.6 1.4 2.9 1.5 3.6 … … … 6.1
… … … … … … … … … …
… … … … … … … … … …
0.6 1.8 2.7 1.9 2.4 2.0 … … … 1.2
0 1 0 0 0 0 0 0 … 0
2.4 2.6 … … 1.8
x_{cat}
V-dim
x_{on}
V-dim
$W^T_{V\times N} \times x_{cat} = v_{cat}$
+
$W^T_{V\times N} \times x_{on} = v_{on}$
$\hat{v} = \frac{v_{cat} + v_{on}}{2}$
Hidden layer
N-dim
Output layer
sat
V-dim



$W_{V\times N}^T \times x_{on} = v_{on}$
Input layer
0.1 2.4 1.6 1.8 0.5 0.9 … … … 3.2
0.5 2.6 1.4 2.9 1.5 3.6 … … … 6.1
0.6 1.8 2.7 1.9 2.4 2.0 … … … 1.2
1.8 2.9 … … 1.9
x_{cat}
V-dim
$W_{V\times N}^T \times x_{cat} = v_{cat}$
+
$W_{V\times N}^T \times x_{on} = v_{on}$
x_{on}
V-dim
$\hat{v} = \frac{v_{cat} + v_{on}}{2}$
Hidden layer
N-dim
Output layer
sat
V-dim


Input layer
cat
V-dim
on
V-dim
$W_{V\times N}$
$W_{V\times N}$
Hidden layer
$\hat{v}$
N-dim
$W'_{V\times N} \times \hat{v} = z$
Output layer
$\hat{y} = softmax(z)$
$\hat{y}_{sat}$
V-dim
N will be the size of word vector


Input layer
cat
V-dim
on
V-dim
$W_{V\times N}$
$W_{V\times N}$
Hidden layer
$\hat{v}$
N-dim
We would prefer $\hat{y}$ close to $\hat{y}_{sat}$
Output layer
$W'_{V\times N} \times \hat{v} = z$
$\hat{y} = softmax(z)$
$\hat{y}_{sat}$
V-dim
0.01
0.02
0.00
0.02
0.01
0.02
0.01
0.7
...
0.00
$\hat{y}$
N will be the size of word vector



0.1	2.4	1.6	1.8	0.5	0.9	…	…	…	3.2
0.5	2.6	1.4	2.9	1.5	3.6	…	…	…	6.1
…	…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…	…
0.6	1.8	2.7	1.9	2.4	2.0	…	…	…	1.2

We can consider either W or W' as the word's representation. Or even take the average.

Some interesting results

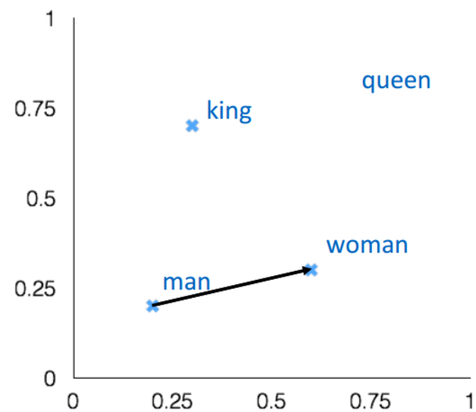
Word Analogies

Test for linear relationships, examined by Mikolov et al. (2014)

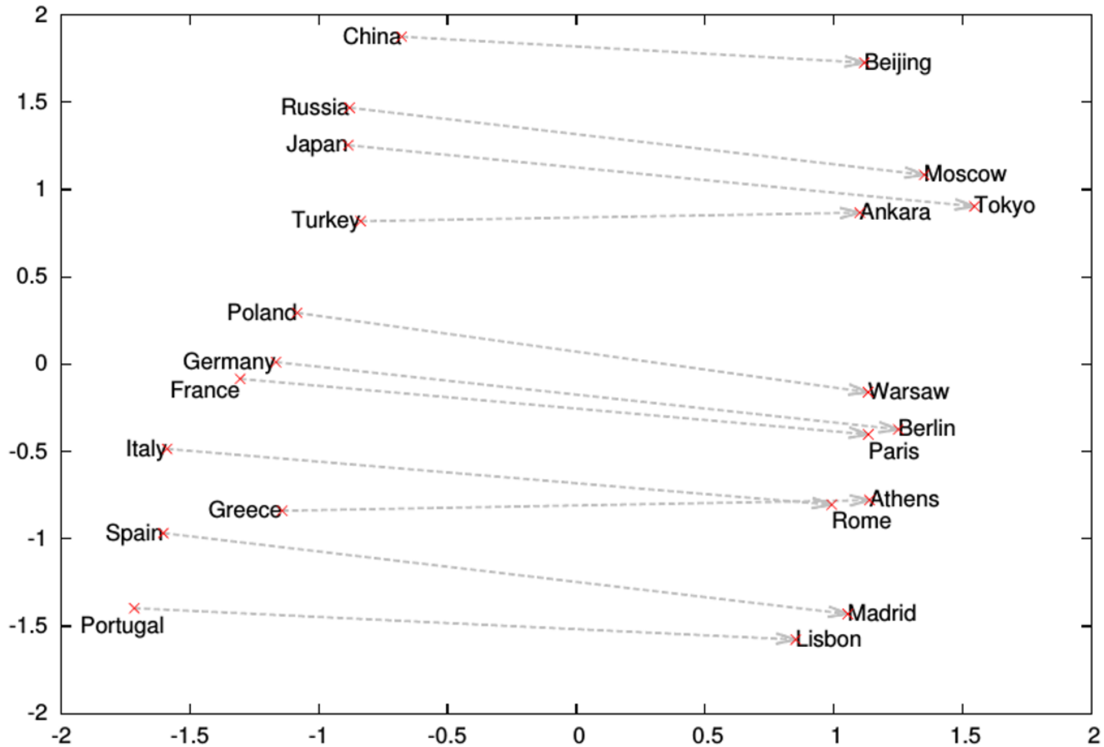
a:b :: c:? → $d = \arg\max_{x} \frac{(w_b - w_a + w_c)^T w_x}{||w_b - w_a + w_c||}$

man:woman :: king:?

+	king	[0.30 0.70]
-	man	[0.20 0.20]
+	woman	[0.60 0.30]
	queen	[0.70 0.80]

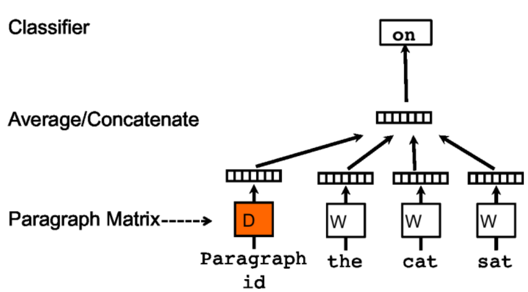


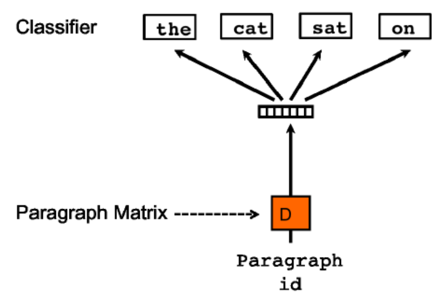
Word analogies



Represent the meaning of **sentence/text**

- Simple approach: take avg of the word2vecs of its words
- Another approach: Paragraph vector (2014, Quoc Le, Mikolov)
  - Extend word2vec to text level
  - Also two models: add paragraph vector as the input





Applications

- Word Similarity: Edit Distance, WordNet, Porter's Stemmer, Lemmatization using dictionaries
- Search, e.g., query expansion
- Machine Translation
- Part-of-Speech and Named Entity Recognition
- Relation extraction
- Sentiment analysis
- Semantic Analysis of Documents
- Clustering