NATURAL LANGUAGE PROCESSING

WITH DEEP LEARNING

Lecturer: Javad PourMostafa

NLP981

Vector Semantics and Embeddings

GOAL

- **Build a new model of meaning focus on similarity**
- Represent words (docs) by vectors

Fact: Similar words will have similar vectors!

- Usage?

  E.g.

  - Question Answering:
  - Q: "How **tall** is Mr. Travers?"
  - Candidate A: "The official **height** of Joe Travers is 186CM"

WORD SIMILARITIES

- While words don't have many synonyms, most words have lots of similar words.
- E.g. Cats and dogs
- Similar words are "nearby in space".

vanish	disappear	9.8
behave	obey	7.3
belief	impression	5.95
muscle	bone	3.65
modest	flexible	0.98
hole	agreement	0.3

EMBEDDINGS

- Called an "**embedding**" because it's embedded into a space
- The standard way to represent meaning in NLP
- 2 types of embedding:

Tf-idf:

- **A Common baseline**
- **Sparse vectors**
- **Represented by a simple function of the count and inverse document frequency**

Word2vec:

- **Dense vectors**
- **Represented is constructed by training a classifier**

Review: Words, vectors, and co-occurrence matrices

TERM-DOCUMENT MATRIX

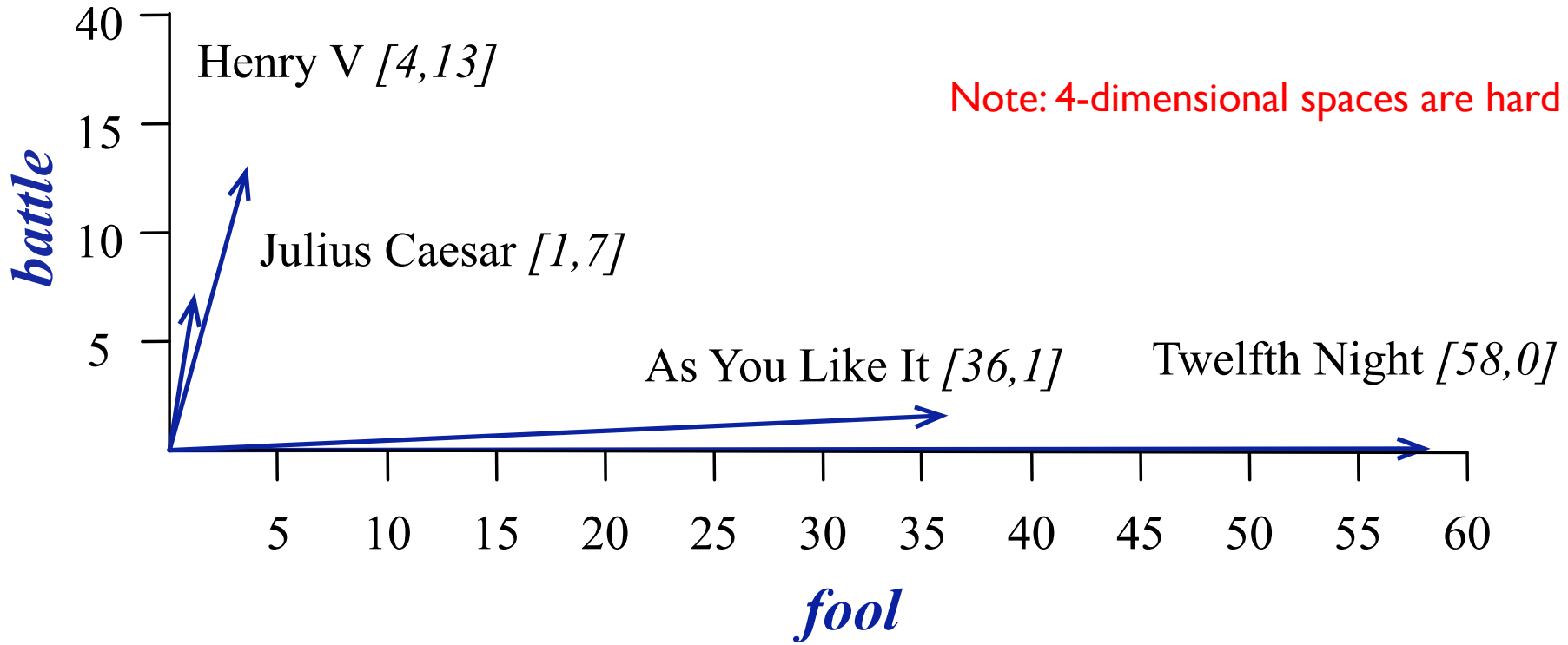
- Each document is represented by a vector of words

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

- Note: Vocabulary set can stand in columns! (word vectors)
- Two comedies look a lot more like each other! (As you like it and Twelfth Night)

VISUALIZING DOCUMENT VECTORS

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3



Note: 4-dimensional spaces are hard to draw! So…

VECTORS IN INFORMATION RETRIEVAL (IR)

Information Retrieval (IR)

- Task of finding Doc ***d*** from ***D*** documents in some collection that best matches as query ***q***.
- Represent a query by a vector, also length $|v|$
- **Then Compare query with each vector in document collection**

- **Doing IR will need store and manipulate vectors in efficient ways (sparsity, zeros, ...)**
  - **Tf-idf term weighting**
  - **Cosine similarity metric**

WORD-WORD MATRIX / TERM-CONTEXT MATRIX

- Can you guess its dimension?
  - $|V| \times |V|$
- Each cell:
  - Row: # of times the target
  - Column: # of times the context

Two **words** are similar in meaning if their context vectors are similar

- Context = a **word window** (e. g. ± 7) around the row word (Previously used whole words in context)
- Example:

sugar, a sliced lemon, a tablespoonful of	**apricot**	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

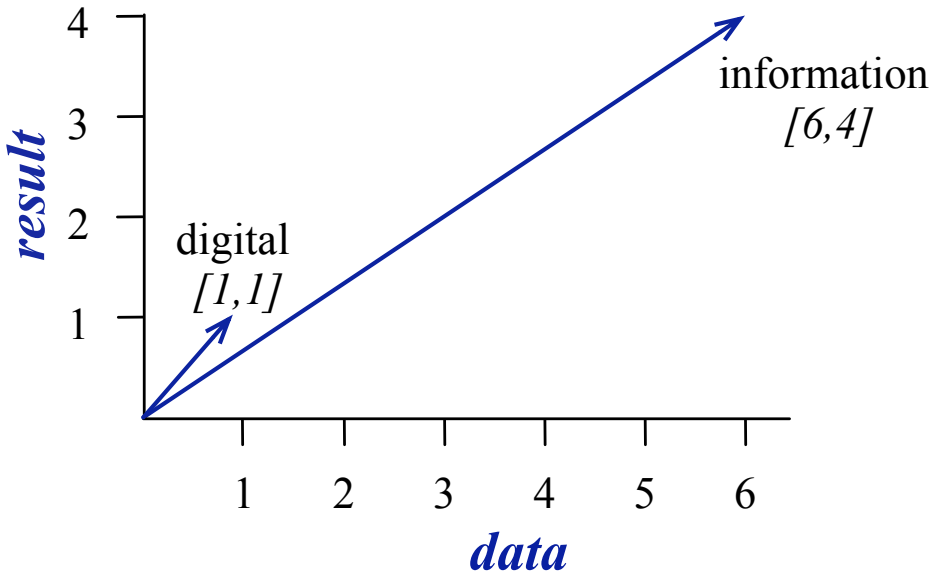
WORD-WORD CO-OCCURRENCE MATRIX

	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	

- Note: Simplified context word collected from Wikipedia corpus
- Columns are filled with most frequent words in corpus
- Still the length of vector is between 10000 and 50000 words) => **Sparse**

WORD-WORD CO-OCCURRENCE MATRIX

	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	



REMINDERS FROM LINEAR ALGEBRA

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

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

Note: dot product also called inner product.

When does a dot-product maximize?

Note: dot product tends to be high just when the two vectors that have large values in same directions.

Note: dot product is higher if a vector is longer!

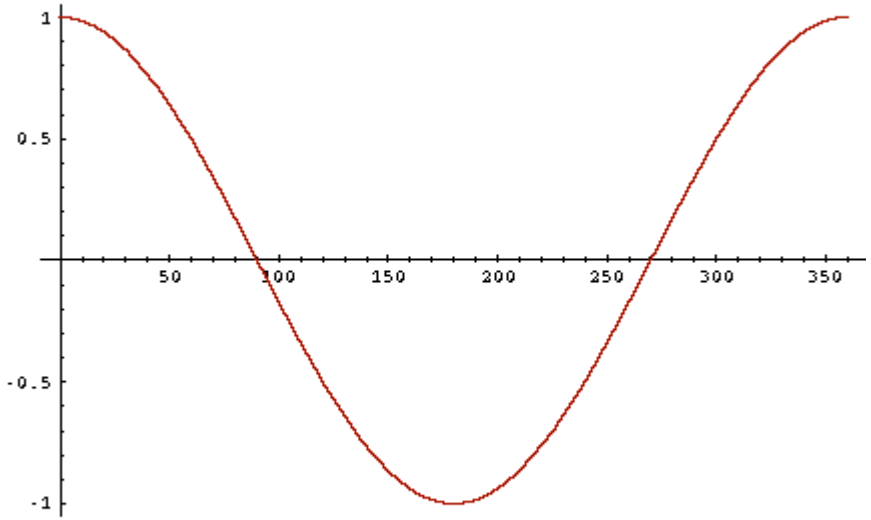
COSINE FOR COMPUTING SIMILARITY

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

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

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

- v_i is the count for word *v* in context *i*
- w_i is the count for word *w* in context *i.*



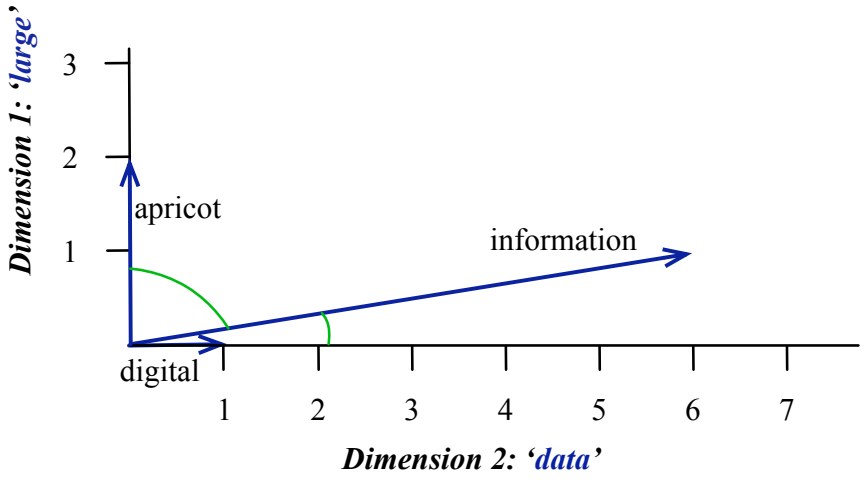
AN EXAMPLE OF COSINE SIMILARITY FOR TERM-CONTEXT MATRIX

Which pair of words is more similar?

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

	large	data	computer
apricot	1	0	0
digital	0	1	2
information	1	6	1

- cosine(apricot, information) = $\frac{1+0+0}{\sqrt{1+0+0}\ \sqrt{1+36+1}} = \frac{1}{\sqrt{38}} = .16$
- cosine(digital, information) = $\frac{0+6+2}{\sqrt{0+1+4}\ \sqrt{1+36+1}} = \frac{8}{\sqrt{38}\sqrt{5}} = .58$
- cosine(apricot, digital) = $\frac{0+0+0}{\sqrt{1+0+0}\ \sqrt{0+1+4}} = 0$



HIGH FREQUENCY IS GOOD BUT NOT ALWAYS

- Simple frequency **isn't the best measure** of association between words.
- Some words occur frequently with all sorts of words like **prepositions**
- High frequent words are good but in all cases
- Low frequency isn't useful also like **typo**
- **Best choices are medium frequencies**
- **But not all mediums**
  - ***So what?***

TF-IDF ALGORITHM: COMBINE TWO FACTORS

Term Frequency:

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

Inverse document frequency:

$$\mathrm{idf}_i = \log\left(\frac{N}{\mathrm{df}_i}\right)$$

Tf-idf:

$$w_{t,d} = \mathrm{tf}_{t,d} \times \mathrm{idf}_t$$

TF-IDF: EXAMPLE

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.074	0	0.22	0.28
good	0	0	0	0
fool	0.019	0.021	0.0036	0.0083
wit	0.049	0.044	0.018	0.022

Figure 6.8 A tf-idf weighted term-document matrix for four words in four Shakespeare plays, using the counts in Fig. 6.2. For example the 0.049 value for *wit* in *As You Like It* is the product of $\text{tf} = \log_{10}(20+1) = 1.322$ and $\text{idf} = .037$. Note that the idf weighting has eliminated the importance of the ubiquitous word *good* and vastly reduced the impact of the almost-ubiquitous word *fool*.

POINTWISE MUTUAL INFORMATION (PMI)

- An alternative function to tf-idf

E.g. In morning, pass away

- The most important concept in NLP
- **Measure how often two events x and y occur, compared with what we would expect if they were independent**
- *Do events x and y co-occur more than if they were independent?*

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

- *Do words and context co-occur more than if they were independent?*

$$\text{PMI}(word, context) = \log_2 \frac{P(word, context)}{P(word)P(context)}$$

POSITIVE POINTWISE MUTUAL INFORMATION (PPMI)

- PMI ranges from $-\infty$ to $+\infty$
- But negative values are problematic (When it will happen)?
  - Things are co-occurring **less than** we expect by chance
- Solution?
  - Just replace negative PMI values by **0**
  - **Positive PMI (PPMI) between word1 and word2:**

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

EXAMPLE OF PPMI ON A TERM-CONTEXT MATRIX

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

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

$$pmi_{ij} = \log_2 \frac{p_{ij}}{p_{i*}p_{*j}} \qquad ppmi_{ij} = \begin{cases} pmi_{ij} & \text{if } pmi_{ij} > 0 \\ 0 & \text{otherwise} \end{cases}$$

Matrix F with W rows (words) and C columns (contexts)

f_{ij} is # of times w_i occurs in context c_j

EXAMPLE OF PPMI ON A TERM-CONTEXT MATRIX

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

Count(w,context)

	computer	data	pinch	result	sugar	
apricot	0	0	1	0	1	2
pineapple	0	0	1	0	1	2
digital	2	1	0	1	0	4
information	1	6	0	4	0	11
	3	7	2	5	2	

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

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

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

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

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

	computer	data	pinch	result	sugar	
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	

EXAMPLE OF PPMI ON A TERM-CONTEXT MATRIX

$$pmi_{ij} = \log_2 \frac{p_{ij}}{p_{i*}p_{*j}}$$

	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	

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

Note: (.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	-

INSTEAD OF PREVIOUS REPRESENTATIONS?

- Tf-idf and PPMI vectors are
  - **Long = 2000 to 50000**
  - **Sparse = zero elements**
- Alternative? **Word2vec**
  - **Short (length 50-1000)**
  - **Dense (most elements are non-zero)**

SPARSE VS. DENSE VECTORS

- **Dense** Vectors works better in **machine learning** systems.
- **Dense** vectors may **generalize** better than storing explicit counts
  - Avoid overfitting
- **Dense** vectors capture **synonyms** better than sparse vectors.
- For example: ***car and automobile*** are synonyms but are distinct dimensions
- Finally: Dense vectors work better in practice !

DENSE EMBEDDINGS SOURCES !!!

Word2vec (its original paper has 7982 citations !!!)

- Tomas Mikolov et al.
- https://code.google.com/archive/p/word2vec/

FastText

- Facebook AI research
- https://fasttext.cc/

GloVe

- Jeffrey Penningtopn, Richard Socher, D. Manning
- https://nlp.stanford.edu/projects/glove/

WORD2VEC

Idea: Predict rather than count

- Very fast to train
- Code available on the web
- Popular embedding method
  - Instead of **counting** how often each word *w* occurs near "apricot"
  - Train a classifier on a binary **prediction** task
  - Is *w* likely to show up near "apricot"?
- We don't actually care about this task but we'll take the learned classifier weights as word embeddings

WORD2VEC: AN INSIGHT

Use running text as implicitly supervised training data!

- A word *s* near *apricot*
  - Acts as gold 'correct answer' to the question
  - "Is word *w* likely to show up near *apricot*?"
- No need for hand-labeled supervision
- The idea comes from **neural language modeling**
  - Bengio et al. (2003)
  - Collobert et al. (2011)

WORD2VEC: SKIP-GRAM ALGORITHM

- Word2vec has many options
- Skip-gram with negative sample (SGNS)

Algorithm:

- 1. Treat the target word and a neighboring context word as **positive examples**.
- 2. **Randomly** sample other words in the lexicon to get **negative samples**.
- 3. Use **logistic regression** to train a classifier to **distinguish those two cases**.
- 4. Use the regression **weights as the embeddings**.

WORD2VEC: SKIP-GRAM GOAL

Training sentence:

... lemon, a tablespoon of **apricot** jam a pinch ...

c1 c2 target c3 c4

- Window size = ± 2
- Given a tuple $(t, c) = target, context$
  - $(apricot, jam)$
  - $(apricot, aardvark)$
- Goal: **Return probability that c is a real context word**

$$P(+|t, c)$$
$$P(-|t, c) = 1 - P(+|t, c)$$

HOW TO COMPUTE $p(+|t,c)$

- Model the similarity with dot-product:

$$\boldsymbol{Similarity\,(t,c)\ \propto t.c}$$

- **Problem?** Dot product is not a probability (Neither is cosine)!
- **Solution?**
  - Turning dot product into a probability (how?)
  - **Sigmoid**

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

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

COMPUTE FOR ALL CONTEXT WORDS

- Assume all context words are **independent**
- K is number of context window

$$P(+|t, c_{1:k}) = \prod_{i=1}^{\kappa} \frac{1}{1+e^{-t \cdot c_i}}$$

$$\log P(+|t, c_{1:k}) = \sum_{i=1}^{k} \log \frac{1}{1+e^{-t \cdot c_i}}$$

SKIP-GRAM TRAINING

- Training sentence:

... lemon, a tablespoon of **apricot** jam a pinch ...

c1 c2 target c3 c4

positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

negative examples -

t	c	t	c
apricot	aardvark	apricot	twelve
apricot	puddle	apricot	hello
apricot	where	apricot	dear
apricot	coaxial	apricot	forever

- For each positive example, we'll create *k* negative examples.
- Using *noise* words
- Any random word that isn't *t*