CS447: Natural Language Processing

*http://courses.engr.illinois.edu/cs447*

# Lecture 6: Vector Semantics and Word Embeddings

Julia Hockenmaier

*juliahmr@illinois.edu*

3324 Siebel Center

# Lecture 6

## Part 1: Lexical Semantics and the Distributional Hypothesis

# Let’s look at words again….

So far, we’ve looked at…
… the **structure** of words (**morphology**)
… the **distribution** of words (**language modeling**)

Today, we’ll start looking at the **meaning** of words (**lexical semantics**).

We will consider:
… the **distributional hypothesis** as a way to identify words with similar meanings
… two kinds of **vector representations** of words that are inspired by the distributional hypothesis

# Today's lecture

Part 1: Lexical Semantics
and the Distributional Hypothesis

Part 2: Distributional similarities
(from words to sparse vectors)

Part 3: Word embeddings
(from words to dense vectors)

Reading: Chapter 6, Jurafsky and Martin (3rd ed).

# What do words **mean**, and how do we **represent** that?


Do we want to represent that…

… "cassoulet" is a French dish?

… "cassoulet" contains meat?

… "cassoulet" is a stew?

# What do words mean, and how do we represent that?


Do we want to represent…

… that a “bar” are places to have a drink?

… that a “bar” is a long rods?

… that to “bar” something means to block it?

# Different approaches to lexical semantics

Roughly speaking, NLP draws on two different types of approaches to capture the meaning of words:

**The lexicographic tradition** aims to capture the information represented in lexicons, dictionaries, etc.

**The distributional tradition** aims to capture the meaning of words based on large amounts of raw text

# The lexicographic tradition

Uses resources such as lexicons, thesauri, ontologies etc. that capture explicit knowledge about word meanings.

Assumes words have *discrete* word senses:

bank1 = financial institution; bank2 = river bank, etc.

May capture *explicit relations* between word (senses):
"*dog*" is a "*mammal*", "*cars*" have "*wheels*" etc.

[ We will talk about this in Lecture 20.]

# The Distributional Tradition

Uses large corpora of raw text to learn the meaning of words from the contexts in which they occur.

Maps words to (sparse) vectors that capture corpus statistics

Contemporary variant: use neural nets to learn dense vector "embeddings" from very large corpora

(this is a prerequisite for most neural approaches to NLP)

If each word type is mapped to a single vector, this ignores the fact that words have multiple senses or parts-of-speech

[Today's class]

# Language understanding requires knowing when words have similar meanings

**Question answering:**

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

*“tall”* is similar to *“height”*

# Language understanding requires knowing when words have similar meanings

## 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

**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

# How do we represent words to capture word similarities?

As **atomic symbols**?

[e.g. as in a traditional n-gram language model, or when we use them as explicit features in a classifier]

This is equivalent to very high-dimensional **one-hot vectors:**

*aardvark*=[1,0,…,0], *bear*=[0,1,000],…, *zebra*=[0,…,0,1]

No: *height/tall* are as different as *height/cat*

As very high-dimensional **sparse vectors**?

[to capture so-called distributional similarities]

As lower-dimensional **dense vectors**?

["word embeddings" — important prerequisite for neural NLP]

# What should word representations capture?

Vector representations of words were originally motivated by attempts to capture lexical semantics (the meaning of words) so that words that have **similar meanings** have **similar representations**

These representations may also capture some **morphological** or **syntactic** properties of words (parts of speech, inflections, stems etc.).

# The Distributional Hypothesis

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."*

John R. Firth 1957:

*You shall know a word by the company it keeps.*

The **contexts** in which a word appears tells us a lot about what it means.

Words that appear in similar contexts have similar meanings

# Why do we care about word contexts?

***What is tezgüino?***

A bottle of **tezgüino** is on the table.
Everybody likes **tezgüino**.
**Tezgüino** makes you drunk.
We make **tezgüino** out of corn.
(Lin, 1998; Nida, 1975)

*Corpus*

A bottle of **wine** is on the table.
There is a **beer** bottle on the table
**Beer** makes you drunk.
We make **bourbon** out of corn.
Everybody likes **chocolate**
Everybody likes **babies**

We don't know exactly what tezgüino is, but since we understand these sentences, it's likely an alcoholic drink.

Could we *automatically* identify that tezgüino is like beer?

A large corpus may contain sentences such as:

**Beer** makes you drunk

But there are also red herrings:

Everybody likes **chocolate**    Everybody likes **babies**

# Two ways NLP uses context for semantics

**Distributional similarities** (vector-space semantics):
Use the set of all contexts in which words (= word types) appear to measure their similarity

Assumption: Words that appear in similar contexts (*tea, coffee*) have similar meanings.

**Word sense disambiguation** (future lecture)
Use the context of a *particular occurrence* of a word (token) to identify which sense it has.

Assumption: If a word has multiple distinct senses (e.g. *plant*: *factory* or *green plant*), each sense will appear in different contexts.

# Lecture 6

## Part 2: Distributional Similarities (From Words to Sparse Vectors)

# Distributional Similarities

**Basic idea:**

Measure the semantic **similarity of words** in terms of the **similarity of the contexts** in which they appear

**How?**

Represent words as **vectors** such that

— each **vector element** (dimension) corresponds to a different **context**

— the vector for any particular word captures how **strongly it is associated** with each context

Compute the semantic similarity of words as the **similarity of their vectors**.

# Distributional similarities

Distributional similarities use the set of contexts in which words appear to measure their similarity.

They represent each word *w* as a vector $\mathbf{w}$

$$\mathbf{w} = (w_1, \ldots, w_N) \in \mathbf{R}^N$$

in an N-dimensional vector space.

- Each dimension corresponds to a particular context $c_n$
- Each element $w_n$ of $\mathbf{w}$ captures the degree to which the word *w* is associated with the context $c_n$.
- $w_n$ depends on the co-occurrence counts of *w* and $c_n$

The similarity of words *w* and *u* is given by the similarity of their vectors $\mathbf{w}$ and $\mathbf{u}$

# The Information Retrieval perspective: The Term-Document Matrix

In IR, we search a **collection of *N* documents**

- We can represent each **word** in the vocabulary *V* as an *N*-dim. **vector** indicating which documents it appears in.
- Conversely, we can represent each **document** as a *V*-dimensional **vector** indicating which words appear in it.

**Finding the most relevant document for a query:**

- Queries are also (short) documents
- Use the similarity of a query's vector and the documents' vectors to compute which document is most relevant to the query.

**Intuition:** Documents are similar to each other if they contain the same words.

# Term-Document Matrix

| | 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 |

A Term-Document Matrix is a 2D table:

- Each **cell** contains the frequency (count) of the term (word) *t* in document *d*: $tf_{t,d}$
- Each **column** is a vector of counts over words, representing a document
- Each **row** is a vector of counts over documents, representing a word

# Term-Document Matrix

| | 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 |

Each column vector = a document

- Each entry corresponds to one word in the vocabulary

Each row vector = a word

- Each entry corresponds to one document in the corpus

Two documents are similar if their vectors are similar

Two words are similar if their vectors are similar

# Now back to lexical semantics

For information retrieval, the term-document matrix is useful because it can be used to compute the similarity of documents in terms of the words they contain, or of words in terms of the documents in which they appear.

But we can adapt this approach to implement a model of the distributional hypothesis if we treat each context as a column in our matrix.

# What is a ‘context’?

There are many different definitions of context that yield different kinds of similarities:

## Contexts defined by nearby **words**:

How often does *w* appear near the word *drink*?

Near = “*drink* appears within a window of ±k words of *w*”, or “*drink* appears in the same document/sentence as *w*”

This yields fairly broad thematic similarities.

## Contexts defined by **grammatical relations**:

How often is (the noun) *w* used as the subject (object) of the verb *drink*? (Requires a parser).

This gives more fine-grained similarities.

# Using nearby words as contexts

Define a **fixed vocabulary** of $N$ **context** words $c_1, \ldots, c_N$

Context words should occur frequently enough in your corpus that you get reliable co-occurrence counts, but you should ignore words that are too common ('stop words': *a*, *the, on, in, and, or, is, have,* etc.)

Define what '**nearby**' means

For example: $w$ appears near $c$ *if* $c$ appears within ±5 words of $w$

Get **co-occurrence counts** of words $w$ and contexts $c$

Define how to transform co-occurrence counts
of words $w$ and contexts $c$ into **vector elements** $w_n$

For example: compute (positive) **PMI** of words and contexts

Define how to compute the **similarity of word vectors**

For example: use the cosine of their angles.

# Word-Word Matrix

## Context: ± 7 words

| | | |
|---|---|---|
| sugar, a sliced lemon, a tablespoonful of | **lemon** | 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 |

## Resulting word-word matrix:

$f(w, c)$ = how often does word $w$ appear in context $c$:

"*information*" appeared six times in the context of "*data*"

| | 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 | |

# Defining and representing co-occurrence of words and contexts

## **Defining** co-occurrences:

- **Within a fixed window**: $v_i$ occurs within $\pm n$ words of $w$
- **Within the same sentence:** requires sentence boundaries
- **By grammatical relations**:
  $v_i$ occurs as a subject/object/modifier/… of verb $w$
  (requires parsing — and separate features for each relation)

## **Representing** co-occurrences:

- $f_i$ as **binary features** (1,0): $w$ does/does not occur with $v_i$
- $f_i$ as **frequencies**: $w$ occurs $n$ times with $v_i$
- $f_i$ as **probabilities**: e.g. $f_i$ is the probability that $v_i$ is the subject of $w$.

# Getting co-occurrence counts

## Co-occurrence as a **binary** feature:

Does word $w$ ever appear in the context $c$? (1 = yes/0 = no)

| | arts | boil | data | function | large | sugar | water |
|---|---|---|---|---|---|---|---|
| **apricot** | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| **pineapple** | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| **digital** | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| **information** | 0 | 0 | 1 | 1 | 1 | 0 | 0 |

## Co-occurrence as a **frequency** count:

How often does word $w$ appear in the context $c$? (0,1,2,… times)

| | arts | boil | data | function | large | sugar | water |
|---|---|---|---|---|---|---|---|
| **apricot** | 0 | 1 | 0 | 0 | 5 | 2 | 7 |
| **pineapple** | 0 | 2 | 0 | 0 | 10 | 8 | 5 |
| **digital** | 0 | 0 | 31 | 8 | 20 | 0 | 0 |
| **information** | 0 | 0 | 35 | 23 | 5 | 0 | 0 |

# Counts vs PMI

Sometimes, low co-occurrences counts are very informative, and high co-occurrence counts are not:

- Any word is going to have relatively high co-occurrence counts with very common contexts (e.g. “it”, “anything”, “is”, etc.), but this won’t tell us much about what that word means.
- We need to identify when co-occurrence counts are higher than we would expect by chance.

We can use **pointwise mutual information (PMI)** values instead of raw frequency counts:

$$PMI(w,c) = \log \frac{p(w,c)}{p(w)p(c)}$$

But this requires us to define $p(\mathrm{w}, \mathrm{c})$, $p(\mathrm{w})$ and $p(\mathrm{c})$

| f(w,c) | | | | | |
|---|---|---|---|---|---|
| | **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 |

$p(w{=}\textbf{information}, c{=}\textbf{data}) = 6/19 = .32$

$p(w{=}\textbf{information}) = 11/19 = .58$

$p(c{=}\textbf{data}) = 7/19 = .37$

$$p(w_i, c_j) = \frac{f(w_i, c_j)}{\sum_{i=1}^{W} \sum_{j=1}^{C} f(w_i, c_j)}$$

$$p(w_i) = \frac{f(w_i)}{N}$$

$$p(c_j) = \frac{f(c_j)}{N}$$

| f(w,c) | | | | | | **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(c)** | 0.16 | 0.37 | 0.11 | 0.26 | 0.11 | |

# Computing PMI of $w$ and $c$: Using a fixed window of $\pm k$ words

| | |
|---|---|
| $N$: | How many tokens does the corpus contain? |
| $f(w) \leq N$: | How often does $w$ occur? |
| $f(w, c) \leq f(w)$ | How often does $w$ occur with c in its window? |
| $f(c) = \Sigma_w f(w, c)$: | How many tokens have c in their window? |

$$p(w) = \frac{f(w)}{N} \qquad p(c) = \frac{f(c)}{N} \qquad p(w, c) = \frac{f(w, c)}{N}$$

$$PMI(w, c) = \log \frac{p(w, c)}{p(w)p(c)}$$

# Computing PMI of $w$ and $c$: $w$ and $c$ in the same sentence

| | |
|---|---|
| $N$: | How many sentences does the corpus contain? |
| $f(w) \leq N$: | How many sentences contain $w$? |
| $f(w,c) \leq f(w)$ | How many sentences contain $w$ and $\mathrm{c}$? |
| $f(c) \leq N$: | How many sentences contain $\mathrm{c}$? |

$$p(w) = \frac{f(w)}{N} \qquad p(c) = \frac{f(c)}{N} \qquad p(w,c) = \frac{f(w,c)}{N}$$

$$PMI(w,c) = \log \frac{p(w,c)}{p(w)p(c)}$$

# Positive Pointwise Mutual Information

PMI is negative when words co-occur less than expected by chance.

This is unreliable without huge corpora:

With $P(w_1) \approx P(w_2) \approx 10^{-6}$, we can't estimate whether $P(w_1, w_2)$ is significantly different from $10^{-12}$

We often just use positive PMI values, and replace all negative PMI values with 0:

**Positive Pointwise Mutual Information** (PPMI):

$$
\begin{aligned}
PPMI(w,c) &= PMI && \text{if } PMI(w,c) > 0 \\
&= 0 && \text{if } PMI(w,c) \leq 0
\end{aligned}
$$

# PMI and smoothing

PMI is biased towards infrequent events:

If $P(w,c) = P(w) = P(c)$, then $PMI(w,c) = \log\left(\frac{1}{P(w)}\right)$

So $PMI(w,c)$ is larger for rare words $\mathrm{w}$ with low $P(w)$.

Simple remedy: Add-k smoothing of $P(\mathrm{w},\mathrm{c}), P(\mathrm{w}), P(\mathrm{c})$ pushes all PMI values towards zero.

Add-k smoothing affects low-probability events more, and will therefore reduce the bias of PMI towards infrequent events.

(Pantel & Turney 2010)

# Dot product as similarity

If the vectors consist of simple binary features (0,1), we can use the **dot product** as **similarity metric**:

$$sim_{dot-prod}(\vec{x}, \vec{y}) = \sum_{i=1}^{N} x_i \times y_i$$

The dot product is a bad metric if the vector elements are arbitrary features: it prefers **long** vectors

If one $x_i$ is very large (and $y_i$ nonzero), $sim(\mathbf{x}, \mathbf{y})$ gets very large
If the number of nonzero $x_i$ and $y_i$ is very large, $sim(\mathbf{x}, \mathbf{y})$ gets very large.
Both can happen with frequent words.

$$\text{length of } \vec{x} : |\vec{x}| = \sqrt{\sum_{i=1}^{N} x_i^2}$$

# Vector similarity: Cosine

One way to define the similarity of two vectors is to use the cosine of their angle.

The cosine of two vectors is their dot product, divided by the product of their lengths:

$$\text{sim}_{cos}(\vec{x}, \vec{y}) = \frac{\sum_{i=1}^{N} x_i \times y_i}{\sqrt{\sum_{i=1}^{N} x_i^2}\sqrt{\sum_{i=1}^{N} y_i^2}} = \frac{\vec{x} \cdot \vec{y}}{|\vec{x}||\vec{y}|}$$

sim(**w**, **u**) = 1: **w** and **u** point in the same direction
sim(**w**, **u**) = 0: **w** and **u** are orthogonal
sim(**w**, **u**) = –1: **w** and **u** point in the opposite direction

# Lecture 6

Part 3:
Word Embeddings
(From Words to Dense Vectors)

# (Static) Word Embeddings

A **(static) word embedding** is a function that maps each word type to a single vector

— These vectors are typically **dense** and have much **lower dimensionality** than the size of the vocabulary

— This mapping function typically ignores that the same string of letters may have **different senses** (*dining table* vs. *a table of contents)* **or parts of speech** (*to table a motion* vs. *a table*)

— This mapping function typically assumes a **fixed size vocabulary** (so an UNK token is still required)

# Word2Vec (Mikolov et al. 2013)

The first really influential dense word embeddings

Two ways to think about Word2Vec:

— a simplification of neural language models

— a binary logistic regression classifier

Variants of Word2Vec

— Two different context representations: CBOW or Skip-Gram

— Two different optimization objectives:
Negative sampling (NS) or hierarchical softmax

# Word2Vec Embeddings

**Main idea:**

Use a binary classifier to predict which words appear in the context of (i.e. near) a target word.

The parameters of that classifier provide a dense vector representation of the target word (embedding)

Words that appear in similar contexts (that have high distributional similarity) will have very similar vector representations.

These models can be trained on large amounts of raw text (and pre-trained embeddings can be downloaded)

# Skip-Gram with negative sampling

Train a binary classifier that decides whether a target word *t* appears in the context of other words $c_{1..k}$

— **Context**: the set of k words near (surrounding) *t*

— Treat the target word *t* and any word that *actually* appears in its context in a real corpus as **positive** examples

— Treat the target word *t* and *randomly sampled* words that don't appear in its context as **negative** examples

— Train a **binary logistic regression** classifier to distinguish these cases

— The **weights** of this classifier depend on the **similarity** of t and the words in $c_{1..k}$

Use the weights of this classifier as embeddings for *t*

# Skip-Gram Goal

Given a tuple $(t, c)$ = target, context

(*apricot*, *jam*)

(*apricot*, *aardvark*)

where some context words $c$ are from real data (*jam*) and others (*aardvark*) are randomly sampled from the vocabulary…

… decide whether $c$ is a real context word for the target $t$ (a positive example):

$c$ is real if

$$P(D=1 \mid t, c) \quad > \quad P(D=0 \mid t, c) = 1 - P(D=1 \mid t, c)$$

# How to compute $P(D = + \mid t, c)$?

**Intuition:**

Words are likely to appear near similar words

**Idea:**

Model similarity with a dot-product of vectors:

$$\text{Similarity}(t, c) = f(\mathbf{tc})$$

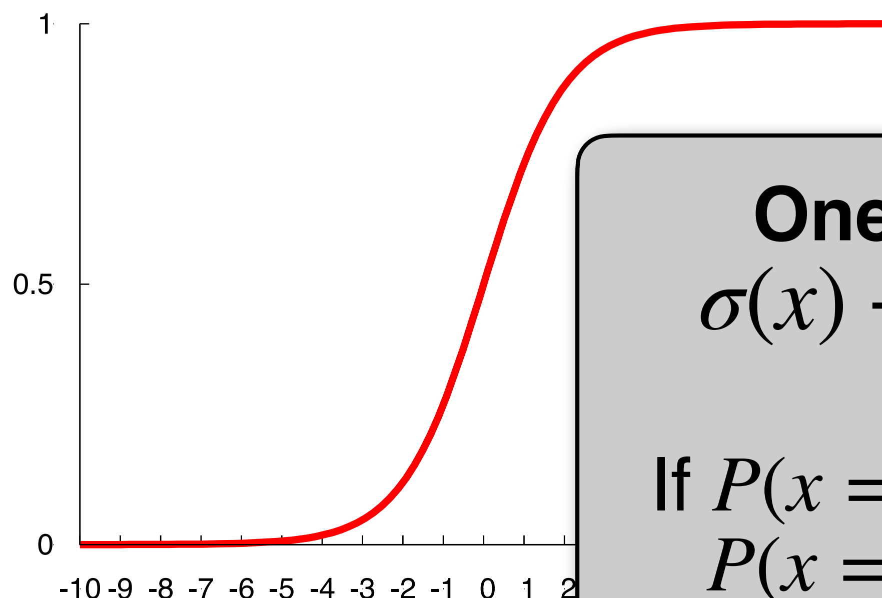
**Problem:**

*The dot product is not a probability!*
*(Neither is cosine)*

# The sigmoid function $\sigma(x)$

The **sigmoid function** $\sigma(x)$ maps
any real number $x$ to the range (0,1):

$$\sigma(x) = \frac{e^x}{e^x + 1} = \frac{1}{1 + e^{-x}}$$




**One more fact:**

$$\sigma(x) + \sigma(-x) = 1$$

If $P(x = \textbf{heads}) = \sigma(x)$,
$P(x = \textbf{tail}) = \sigma(-x)$

# Skip-Gram Training data

**Training sentence:**

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

c1 c2 t c3 c4

**Training data:** input/output pairs centering on *apricot*

Assume a +/- 2 word window

**Positive examples (D+):**

(apricot, tablespoon), (apricot, of), (apricot, jam), (apricot, a)

**Negative examples (D-):**

(apricot, aardvark), (apricot, puddle)…

for each positive example, sample *k* ***noise words***

# Sampling negative examples

Where do we get D- from?

Lots of options.

Word2Vec: for each good pair $(w, c)$, sample $k$ words and add each $w_i$ as a negative example $(w_i, c)$ to $D'$

($D'$ is k times as large as D)

Words can be sampled according to corpus frequency or according to smoothed variant where $\text{freq}'(w) = \text{freq}(w)^{0.75}$

(This gives more weight to rare words)

# The Skip-Gram classifier

Assume that $t$ and $c$ are represented as **vectors** $\mathbf{t}$, $\mathbf{c}$,
so that their dot product $\mathbf{tc}$ captures their similarity

Use **logistic regression** to predict whether the pair $(t, c)$
(target $t$ and context word $c$), is a positive or negative example:

$$P(+\mid t, c) = \frac{1}{1+e^{-\mathbf{tc}}} = \sigma(\mathbf{tc})$$

high if $\mathbf{t}, \mathbf{c}$ very similar

$$P(-\mid t, c) = \frac{e^{-\mathbf{tc}}}{1+e^{-\mathbf{tc}}} = \sigma(-\mathbf{tc})$$

high if $\mathbf{t}, \mathbf{c}$ very dissimilar

NB: When we discussed logistic regression in the last lecture,
we assumed the model learns weights **w** for the feature vector **x**

Skip-Gram learns two (sets of) vectors (i.e. two matrices):
target embeddings/vectors **t** and context embeddings/vectors **c**

# Training objective

Find a model that maximizes the log-likelihood of the training data D+ ∪ D-:

$$\mathscr{L}(D^+, D^-) = \sum_{(t,c)\in D^+} \log P(+ \mid t, c) + \sum_{(t,c)\in D^-} \log P(- \mid t, c)$$

$$= \sum_{(t,c)\in D^+} \sigma(\mathbf{tc}) + \sum_{(t,c)\in D^-} \sigma(-\mathbf{tc})$$

This forces the target and context embeddings of *positive* examples to be *similar* to each other…

… and the target and context embeddings of *negative* examples to be *dissimilar* to each other.

All words appear with positive and negative contexts.

# Summary: How to learn word2vec (skip-gram) embeddings

For a vocabulary of size V: Start with V **random vectors** (typically 300-dimensional) as **initial embeddings**

Train a **logistic regression classifier** to distinguish words that co-occur in corpus from those that don't

— Pairs of words that co-occur are **positive** examples

— Pairs of words that don't co-occur are **negative** examples

During training, target and context vectors of positive examples will become similar, and those of negative examples will become dissimilar.

This returns two embedding matrices **T** and **C,** where each word in the vocabulary is mapped to a 300-dim. vector.

# Properties of embeddings

Similarity depends on window size C

C = ±2 The nearest words to *Hogwarts:*

*Sunnydale*

*Evernight*

C = ±5 The nearest words to *Hogwarts:*

*Dumbledore*

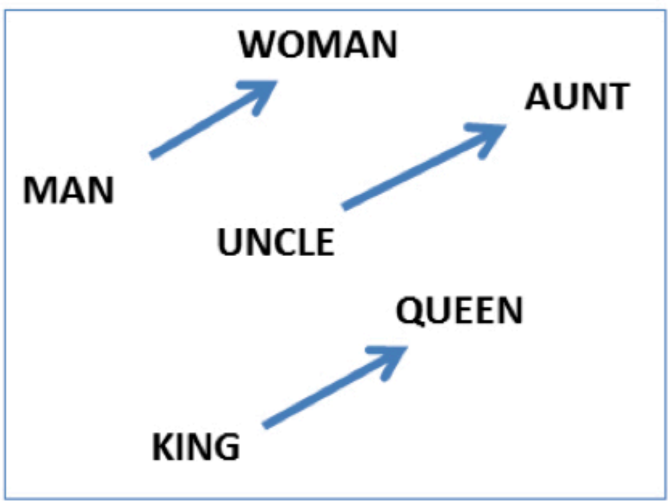
*Malfoy*

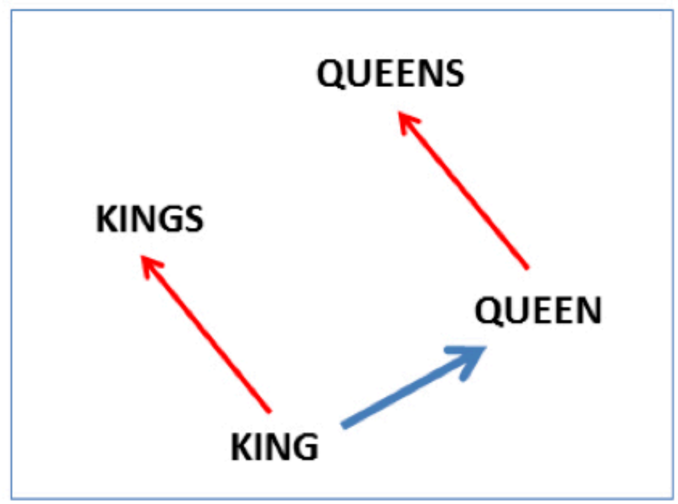
*halfblood*

# Analogy: Embeddings capture relational meaning!

vector(‘*king*’) - vector(‘*man*’) + vector(‘*woman*’) = vector(‘queen’)

vector(‘*Paris*’) - vector(‘*France*’) + vector(‘*Italy*’) = vector(‘*Rome*’)



# Evaluating embeddings

Compare to human scores on word similarity-type tasks:

- WordSim-353 (Finkelstein et al., 2002)
- SimLex-999 (Hill et al., 2015)
- Stanford Contextual Word Similarity (SCWS) dataset (Huang et al., 2012)
- TOEFL dataset: *Levied is closest in meaning to: imposed, believed, requested, correlated*

# Using pre-trained embeddings

Assume you have pre-trained embeddings E.
How do you use them in your model?

- Option 1: Adapt E during training

Disadvantage: only words in training data will be affected.

- Option 2: Keep E fixed, but add another hidden layer that is learned for your task
- Option 3: Learn matrix T $\in$ dim(emb)×dim(emb) and use rows of E' = ET (adapts all embeddings, not specific words)
- Option 4: Keep E fixed, but learn matrix $\Delta \in R^{|V| \times \dim(\text{emb})}$ and use E' = E + Δ or E' = ET + Δ (this learns to adapt specific words)

# Vector representations of words

"Traditional" distributional similarity approaches represent words as sparse vectors

- Each dimension represents one specific context
- Vector entries are based on word-context co-occurrence statistics (counts or PMI values)

Alternative, dense vector representations:

- We can use Singular Value Decomposition to turn these sparse vectors into dense vectors (Latent Semantic Analysis)
- We can also use neural models to explicitly learn a dense vector representation (embedding) (word2vec, Glove, etc.)

Sparse vectors = most entries are zero
Dense vectors = most entries are non-zero

# Dense embeddings you can download!

**Word2vec** (Mikolov et al.)
https://code.google.com/archive/p/word2vec/
**Fasttext** http://www.fasttext.cc/

**Glove** (Pennington, Socher, Manning)
http://nlp.stanford.edu/projects/glove/

# The End