Vector Semantics and Embeddings

- **Lexical Semantics**
- **Vector Semantics**
- **Embeddings: Word2vec**

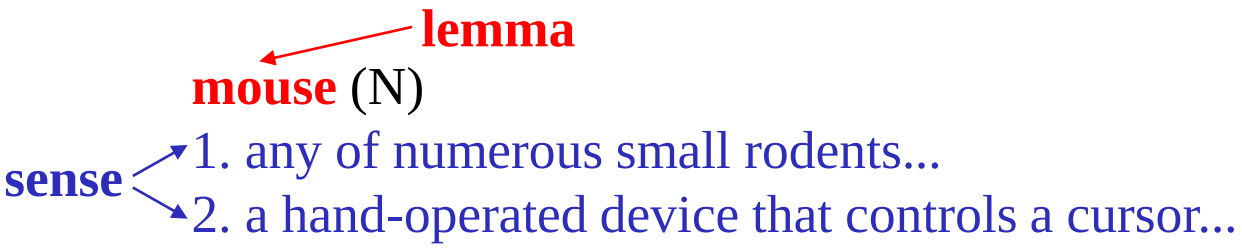
Lexical Semantics

Lexical Semantics

- **How should we represent the meaning of a word?**
  - A model of word meaning should allow us to draw useful inferences that will help us solve meaning-related tasks like question-answering, summarization or plagiarism detection.
- **What do words mean?**
  - Words are just strings (or indices w_i in a vocabulary list) ➔That's not very satisfactory!
  - In logic, the meaning of words can be represented by just spelling the word with small capital letters; representing the meaning of “dog” as DOG, and “cat” as CAT.
    - The meaning of "dog" is DOG; cat is CAT; $\forall x\ \mathrm{DOG}(x) \rightarrow \mathrm{MAMMAL}(x)$
  - **Modern NLP algorithms use embeddings as the representation of word meaning**
- **The linguistic study of word meaning is called Lexical Semantics.**

Words, Lemmas and Senses

- **What's a word?**
  - Types, tokens, stems, roots, inflected forms?
- A word ***mouse*** might be defined in a dictionary as follows:



- The form mouse is the **lemma**.
  - The form mouse would also be the **lemma** for the word mice;
    - dictionaries don't have separate definitions for **inflected forms** like mice.
  - Similarly sing is the lemma for sing, sang, sung.
    - In many languages the **infinitive form** is used as the **lemma for the verb.**
- The ***specific forms*** *sung*, *carpets* or *sing* are called **word forms**.

Words, Lemmas and Senses

- Each lemma can have multiple meanings;
  - the lemma mouse can refer to *rodent* or *cursor control device*.
- We call each of these aspects of the meaning of *mouse* a **word sense**.

- The fact that lemmas can be **homonymous** (have multiple senses) can make interpretation difficult.
  - mouse ➔ *rodent* or *cursor control device*.
  - **Word sense disambiguation** is the task of determining which sense of a word is being used in a particular context.

- There can be many different **relationships between word senses.**
  - synonymy, antonym, similarity, superordinate/subordinate, …

Relation: Synonym

- **When one word has a sense whose meaning is identical to a sense of another word, or nearly identical, we say the two senses of those two words are synonyms.**
  - couch/sofa, big/large, car/automobile, vomit/throw up, …

- Note that there are probably no examples of *perfect synonymy*.
  - Even if many aspects of meaning are identical
  - Still may not preserve the acceptability based on notions of politeness, slang, register, genre, etc.
    - That's my ***big*** sister a ***big*** plane
    - That's my ***large*** sister a ***large*** plane

- **The Linguistic Principle of Contrast:**
  - Difference in form → difference in meaning

Relation: Antonym

- **Two word senses can be antonyms** if they are opposites with respect to one feature of meaning.

`dark/light`	`short/long`	`fast/slow`	`rise/fall`
`hot/cold`	`up/down`	`in/out`	

- **Antonyms** can define a binary opposition with respect to one feature of meaning or are at opposite ends of some scale.
  - `long/short` or `big/little`, which are at opposite ends of the length or size scale.
  - **Reversives** are another group of antonyms which describe change or movement in opposite directions, such as `rise/fall` or `up/down`.

- Antonyms thus differ completely with respect to one aspect of their meaning but they are otherwise very similar, sharing almost all other aspects of meaning.
  - Thus, automatically distinguishing *synonyms* from *antonyms* can be difficult because they appear in similar contexts.

Relation: Word Similarity

- While words don't have many synonyms, most words do have lots of **similar words**.
  - *Cat* is not a synonym of *dog*, but *cats* and *dogs* are certainly **similar words**.

- **Similar words (words with similar meanings) share some element of meaning.**
  - Although similar words are not synonyms, but they share some element of meaning.
    - *Cat* and *dog* are both animals (house pet).

- Which words are similar?

```
car   suv   bicycle   eggplant
```

- The notion of **word similarity** is very useful in larger semantic tasks.
  - *Word similarity* help to determine *phrase or sentence similarity* .
  - *Phrase or sentence similarity* is useful in NLP tasks as question answering, paraphrasing, and summarization.

Relation: Word Similarity

- One way of getting values for word similarity is to ask humans to judge how similar one word is to another.
  - SimLex-999 dataset (Hill et al., 2015) gives values on a scale from 0 to 10 which range from near-synonyms (*vanish*, *disappear*) to pairs that scarcely seem to have anything in common (*hole*, *agreement*):

word1	word2	similarity
vanish	disappear	9.8
behave	obey	7.3
belief	impression	5.95
muscle	bone	3.65
modest	flexible	0.98
hole	agreement	0.3

Relation: Word Relatedness

- **Word Relatedness: The meaning of two words can be related in ways other than similarity.**

- The word relatedness is also called as **word association**.

- *Coffee* is **not similar** to *cup* because they share practically no features.
- But *coffee* and *cup* **are related** since they are associated in a shared event (the event of drinking coffee out of a cup).

- *Scalpel* and *surgeon* are **not similar** but **are related** eventively (a *surgeon* tends to make use of a *scalpel*).

Relation: Word Relatedness - Semantic Field

- One common kind of relatedness between words is **if they belong to the same semantic field**.
- A **semantic field** is **a set of words which cover a particular semantic domain and bear structured relations with each other**.

Semantic Field Examples:

- Hospitals: surgeon, scalpel, nurse, anesthetic, hospital
- Restaurants: waiter, menu, plate, food, menu, chef,
- Houses: door, roof, kitchen, family, bed

- Semantic fields are also related to **topic models**.
- **Semantic fields** and **topic models** are a very useful tool for *discovering topical structure in documents.*

Relation: Word Relatedness - Semantic Frame

- A **semantic frame** is a set of words that denote perspectives or participants in a particular type of event.
  - A *commercial transaction* is a kind of event in which one entity trades money to another entity in return for some good, after which the good changes hands.
  - The *commercial transaction* event can be encoded lexically by using verbs like **buy** or **sell**.

- **Semantic frames** have **semantic roles** (like *buyer*, *seller*, *goods*), and words in a sentence can take on these roles.

- Sam bought the book from John could be paraphrased as John sold the book to Sam by knowing
  - a commercial transaction can be encoded by **buy** or **sell** .
  - Semantic roles:
    - Buyer: **Sam** Seller: **John** Goods: **book**

Relation: Taxonomic Relations

- A word (or word sense) is a **hyponym (subordinate)** of another word (or word sense) if the first is more *specific*, denoting a *subclass* of the other.

- A word (or word sense) is a **hypernym (superordinate)** of another word (or word sense) if the first is more *general*, denoting a *superclass* of the other.

  - **car** is a **subordinate** of **vehicle** — **vehicle** is a **superordinate** of **car**
  - **mango** is a **subordinate** of **fruit** — **fruit** is a **superordinate** of **mango**

Relation: Taxonomic Relations

- **Hypernymy** can also be defined in terms of **entailment**.
  - A word sense A is a hyponym of a word sense B if everything that is A is also B, and hence being an A entails being a B.

- Another name for the **hypernym/hyponym** structure is the **IS-A hierarchy**, in which we say **A IS-A B**, or **B subsumes A**.
  - **IS-A:** **car IS-A vehicle**
  - **Entailment:** every **car** is a **vehicle**

- **Hypernymy** is useful for tasks like *textual entailment* or *question answering*.
  - knowing that *leukemia* is a type of *cancer* would certainly be useful in answering questions about *leukemia*.

Relation: Connotation

- Words have **affective meanings (**or **connotations**).
  - **connotation** means the aspects of a word's meaning that are related to a writer or reader's emotions, sentiment, opinions, or evaluations.

  - ***positive connotations:*** happy
  - ***negative connotations:*** sad
  - ***positive evaluation:*** great, love
  - ***negative evaluation:*** terrible, hate

- *Positive or negative evaluation expressed through language* is called **sentiment.**

Vector Semantics

Vector Semantics

- How can we build a computational model that successfully deals with the different aspects of word meaning (word senses, word similarity and relatedness, semantic fields and frames, connotation)?
- There is **NO perfect model** that completely deals with each of the different aspects of word meaning.

- One computational model is the usage of a lexicon such as **WordNet**:
  - WordNet is a database of **lexical relations** for English (and other languages).

- **The current best model to represent word meaning in NLP is vector semantics.**

Vector Semantics

Distributionalist Approach

- **Ludwig Wittgenstein** who was skeptical of the possibility of building a completely formal theory of meaning definitions for each word suggested instead that **"the meaning of a word is its use in the language".**
  - That is, instead of using some logical language to define each word, we should define words by some representation of how the word was used by actual people.

- **In distributionalist approach, a word is defined by its environment or its distribution in language use.**
  - **Words are defined by their usages.**

- A **word's distribution** is the set of contexts in which it occurs, the neighboring words or grammatical environments.

- The idea is that two words that occur in very similar distributions (that occur together with very similar words) are likely to have the same meaning.

Vector Semantics

Distributionalist Approach: Example

- We do not know the meaning of the word **ongchoi**, but we see its usage in the following sentences (contexts).
  - **Ongchoi** is delicious sauteed with garlic.
  - **Ongchoi** is superb over rice.
- Some of context words of **ongchoi** (delicious, sauteed, garlic, rice) also occur in the following contexts.
  - ...**spinach** sauteed with garlic over rice...
  - ...**spinach** leaves are delicious...
- The fact that the context words of **ongchoi** also occur around **spinach** can help us discover the similarity between **spinach** and **ongchoi**.
  - **Ongchoi** is **"water spinach"**

Vector Semantics

- Vector semantics combines two intuitions:

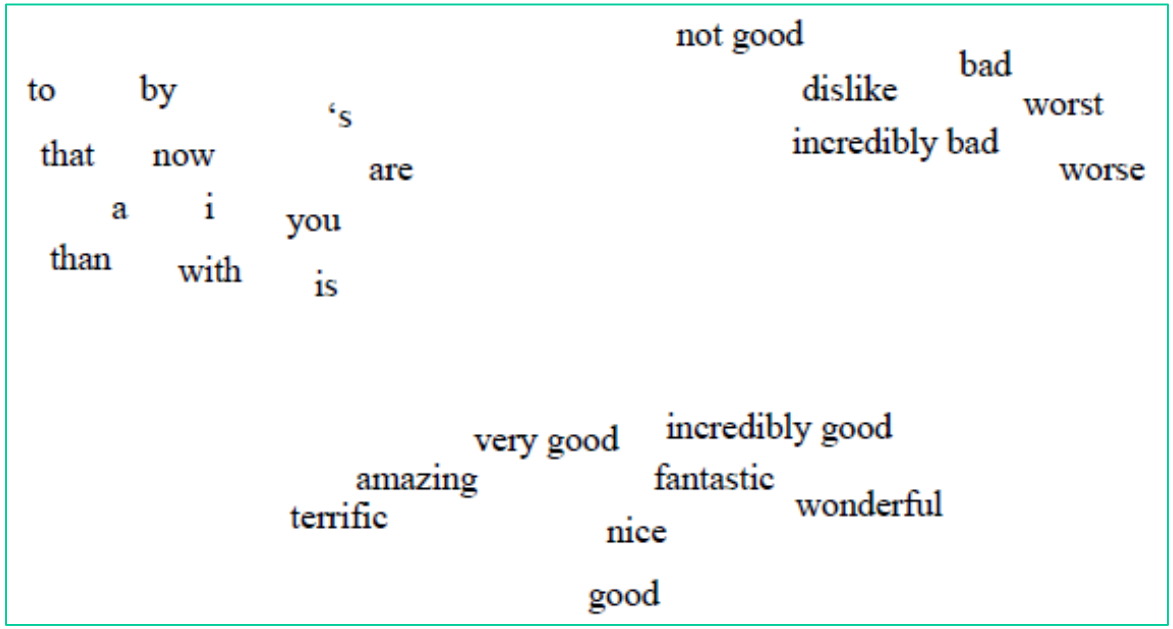
  Distributionalist Intuition: Define a word by counting what other words occur in its environment.

  Vector Intuition: Define the meaning of a word w as a vector, a list of numbers, a point in N dimensional space.

- There are various versions of vector semantics, each defining the numbers in the vector somewhat differently, but in each case the numbers are based in some way on counts of neighboring words.
- The **idea of vector semantics** is ***to represent a word as a point in some multidimensional semantic space.***

Vector Semantics: embeddings

- **Vectors for representing words** are generally called **embeddings**, because the word is embedded in a particular vector space.



positive and negative words seem to be located in distinct portions of the space

A two-dimensional projection of embeddings (learned for a sentiment analysis task) for some words and phrases, showing that words with similar meanings are nearby in space.

Vector Semantics: embeddings

- We define meaning of a word as a **vector,** and those vectors are called as **embeddings.**
- **Every modern NLP algorithm uses embeddings as the representation of word meaning**

- **Embeddings** are a fine-grained model of meaning for similarity.
  - In embeddings, similar words appear "nearby in space".
  - In Sentiment analysis:
    - With words, requires **same** word to be in training and test.
    - With embeddings, ok if **similar** words occurred!!!
- **Vector semantic models** are also extremely practical because they can be learned automatically from text without any complex labeling or supervision.

Vector Semantics: embeddings

- We will look at two **vector semantic models** :

tf-idf model:

- often used a baseline, very long vectors that are sparse,
- the meaning of a word is defined by a simple function of the counts of nearby words.

word2vec model:

- Dense vectors
- Representation is created by training a classifier to distinguish nearby and far-away words.

Vector Semantics: *Words and Vectors*

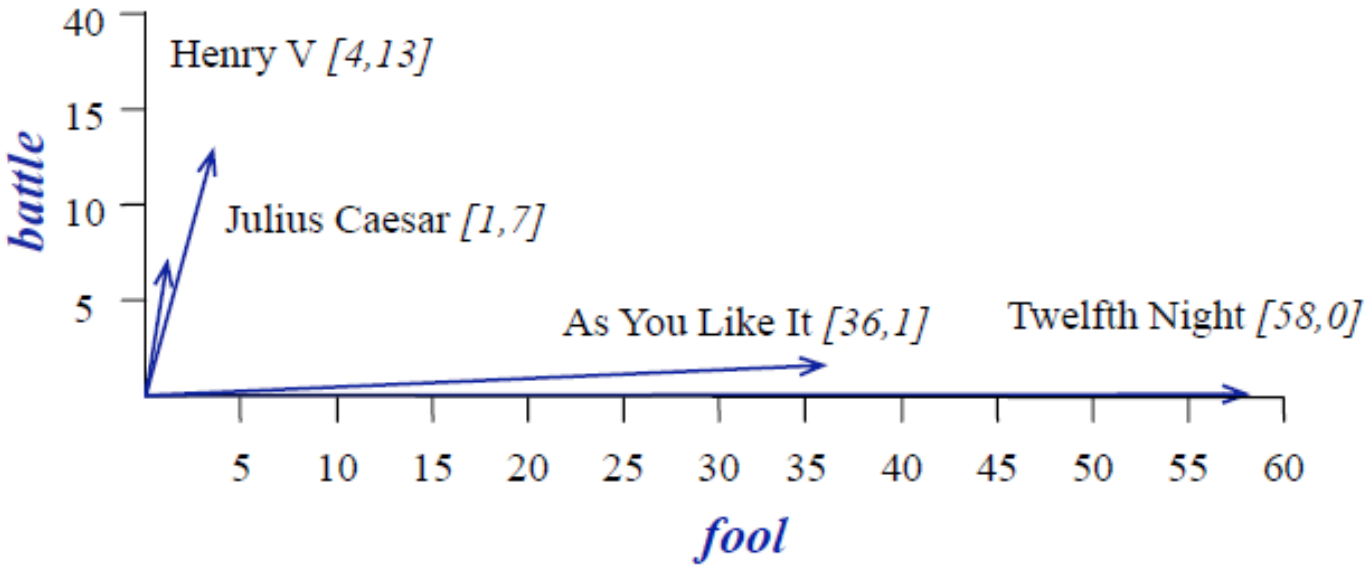
Term-Document Matrix

- Vector models of meaning are generally based on a co-occurrence matrix, a way of representing how often words co-occur.
- In a **term-document matrix**, each row represents a word in the vocabulary and each column represents a document from some collection of documents.
- **Each document is represented by a vector of words.**
  - **The vector for a document is a point in |V|-dimensional space.**

	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

- The term-document matrix for four words in four Shakespeare plays.
- The red boxes show that each document is represented as a column vector of length four.

Visualizing Document Vectors



- A spatial visualization of the document vectors for the four Shakespeare play documents, showing just two of the dimensions, corresponding to the words **battle** and **fool**.
- The comedies have high values for the **fool** dimension and low values for the **battle** dimension.

Term-Document Matrix

- Term-document matrices are used to find similar documents for the task of ***information retrieval***.
  - Two documents that are similar tend to have similar words.

- The vectors of two comedy documents are similar and they are different than the vectors of two history documents.
  - Comedies have more **fools** and **wit** and fewer **battles**.

	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

Words as Vectors

- Words can be vectors too!
- The **word vector** is a row vector rather than a column vector.
  - The four dimensions of the vector for **fool**, **wit**, **battle**, **good**.
  - Each entry in the vector thus represents the counts of the word's occurrence in the document corresponding to that dimension.

	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

  - **battle** is "the kind of word that occurs in ***history*** documents (Julius Caesar and Henry V)"
  - **fool** is "the kind of word that occurs in ***comedies***, especially Twelfth Night"

Word-Word Matrix

- Rather than the term-document matrix, we use the **term-term matrix**, more commonly called the **word-word matrix** (or the ***term-context matrix***) in which the columns are labeled by words rather than documents.
  - **The matrix is |V|x|V| matrix and each cell records number of times the row (target) word and column (context) word co-occur in some context in some training corpus.**
    - The context could be the document, in which case the cell represents the number of times the two words appear in the same document.
    - It is most common to use smaller contexts: generally a window around the word, for example of 4 words to left and 4 words to right, in which case the cell represents number of times column word occurs in such a ± 4 word window around row word.
- Each row in word-word matrix is **co-occurrence vector (context vector) of that row (target) word**.
- Two words are **similar in meaning** if **their context vectors are similar**.

Word-Word Matrix

- Co-occurrence vectors for four words, computed from the Wikipedia corpus, showing only six of the dimensions.
  - A real vector would have vastly more dimensions and thus be much sparser.

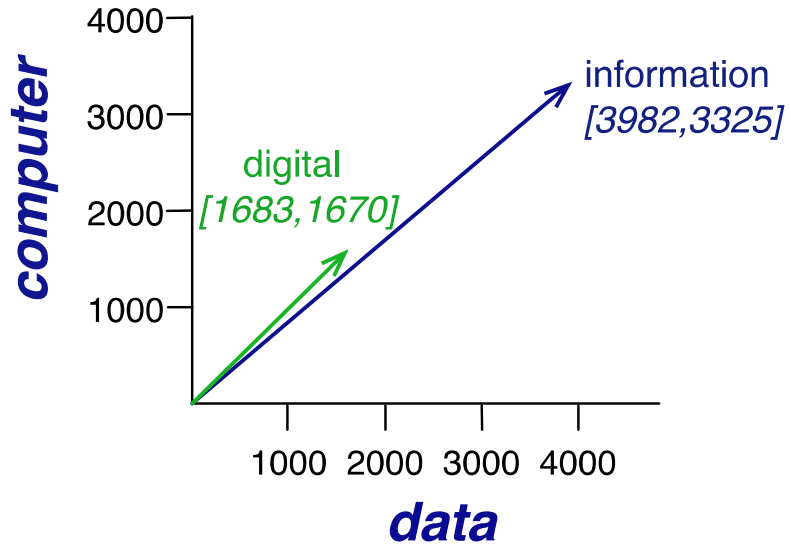
	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...

  - **cherry** and **strawberry** are more similar to each other (both **pie** and **sugar** tend to occur in their window).
  - **digital** and **information** are more similar to each other.

Word-Word Matrix

	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...

- A spatial visualization of word vectors for **digital** and **information**, showing just two of the dimensions, corresponding to the words **data** and **result**.
- In real life these vectors aren't just of length 2. The length of the vector, is generally the size of the vocabulary (more than 50,000).
- Since most of these numbers are zero these are sparse vector representations.



Cosine for Measuring Similarity

- To define similarity between two target words **v** and **w**, we need a measure for taking two such vectors and giving a measure of vector similarity.
- The **most common similarity metric** is the **cosine of the angle between the vectors**.

- The **cosine** is based on **dot product** operator, also called **inner product**:

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

- The ***dot product acts as a similarity metric*** because:
  - It will tend to be high just when the two vectors have large values in the same dimensions.
  - Alternatively, vectors that have zeros in different dimensions—orthogonal vectors—will have a dot product of 0, representing their strong **dissimilarity**.

Cosine for Measuring Similarity

- The **raw dot-product** *has a problem as a similarity metric*: it favors long vectors.

- The **vector length** is defined as: $$|\mathbf{v}| = \sqrt{\sum_{i=1}^{N} v_i^2}$$

  - The dot product is higher if a vector is longer, with higher values in each dimension.
  - More frequent words have longer vectors, since they tend to co-occur with more words and have higher co-occurrence values with each of them.
  - The raw dot product will be higher for frequent words.
  - But this is a problem; we'd like a similarity metric that tells us how similar two words are regardless of their frequency.

Cosine for Measuring Similarity

- The **normalized dot product** is same as **cosine of the angle between two vectors.**
- **Cosine similarity metric between two vectors $\vec{v}$ and $\vec{w}$ can be computed as:**

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

- The cosine value ranges from **1 for vectors pointing in the same direction**, through **0 for vectors that are orthogonal**, *to -1 for vectors pointing in opposite directions*.
- But raw frequency values are non-negative, so cosine for these vectors ranges from 0 to 1.

Cosine for Measuring Similarity

- Cosine computes which of the words **cherry** or **digital** is closer in meaning to **information.**

	pie	data	computer
cherry	442	8	2
digital	5	1683	1670
information	5	3982	3325

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

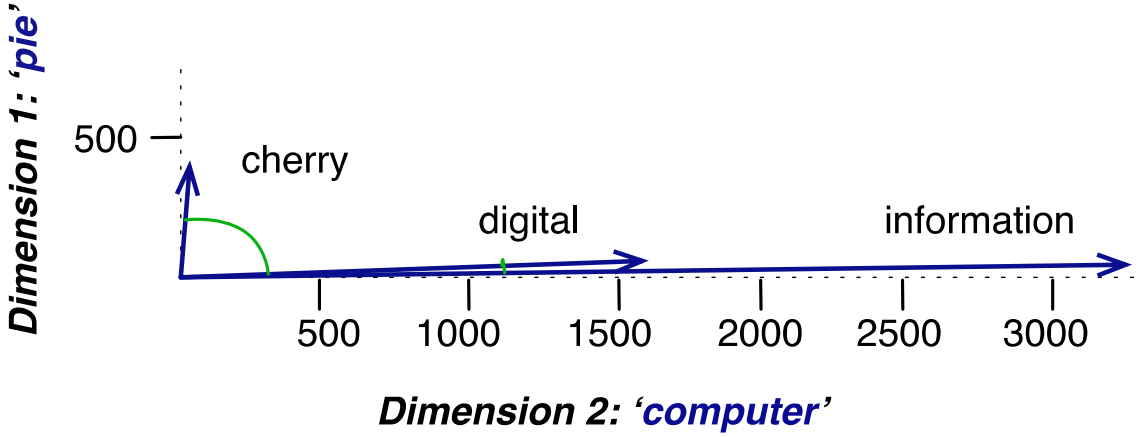
$$\cos(\text{cherry},\text{information}) = \frac{442*5+8*3982+2*3325}{\sqrt{442^2+8^2+2^2}\sqrt{5^2+3982^2+3325^2}} = .018$$

$$\cos(\text{digital},\text{information}) = \frac{5*5+1683*3982+1670*3325}{\sqrt{5^2+1683^2+1670^2}\sqrt{5^2+3982^2+3325^2}} = .996$$

- The model decides that **information** is closer to **digital** than it is to **cherrey.**
  - cos(0) = 1 cos(90) = 0

Cosine for Measuring Similarity

- A graphical demonstration of **cosine similarity**, showing vectors for three words (**cherry**, **digital**, and **information**) in the ***two dimensional space*** defined by counts of the words **pie** and **computer** in the neighborhood.
  - The angle between **digital** and **information** is smaller than the angle between **cherry** and **information**.
  - When ***two vectors are more similar, the cosine is larger but the angle is smaller***



	pie	data	computer
cherry	442	8	2
digital	5	1683	1670
information	5	3982	3325

TF-IDF: Weighing Terms in Vector

- Although frequency is important, but ***simple frequency*** isn't the best measure of association between words.
- **Word that occur nearby frequently** (maybe sugar appears often in our corpus near cherry) **are more important than words that only appear once or twice**.
- **On the other hand, words that are too frequent such as the or a are unimportant.**

- **How can we balance these two conflicting constraints?**
  - **We need a function that resolves this frequency paradox!**

    ➔ **one solution is TF-IDF (term frequency _ inverse document frequency)**
  - **TF-IDF value (the '-' here is a hyphen, not a minus sign) is the product of two terms, each term capturing one of these two intuitions.**

TF-IDF

tf: term frequency: the frequency of the word in the document.

- frequency is down-weighted by using its log value.
  - a word appearing 100 times in a document doesn't make that word 100 times more likely to be relevant to the meaning of the document.

- **term frequency:** $\mathbf{tf_{t,d} = count(t,d)}$
- **term frequency (**instead of using raw count, we can squash a bit)

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

  - count=0➔tf=0; count=1➔tf=1; count=10➔ tf=2; count=100➔tf=3

- **Term frequency** can also be defined as:

  $\mathbf{tf_{t,d} = count(t,d) / \#ofwords(d)}$

TF-IDF

idf: inverse document frequency: give a higher weight to words that occur only in a few documents.

- Terms that are limited to a few documents are useful for discriminating those documents; terms that occur frequently across the entire collection aren't as helpful.
- **Document frequency df_t of a term t is the number of documents it occurs in.**
- **Inverse document frequency idf** can be using the fraction **N/df_t** where N is the number of documents in the collection.
  - Because of the large number of documents in many collections, this measure is usually squashed with a log function.

Inverse document frequency idf_t is defined as:

$$\text{idf}_t = \log_{10}\left(\frac{N}{\text{df}_t}\right)$$

where N is the number of documents in the collection and df_t is the number of documents that contain the term t.

TF-IDF

TF-IDF Value:

- **tf-idf weighting of the value $w_{t,d}$ for term t in document d combines term frequency with idf:**

$$w_{t,d} = tf_{t,d} * idf_t$$

TF-IDF – Example

- A collection contains 37 documents (Shakespeare's play):
- **idf values** of some words in this collection:

Word	df	idf
Romeo	1	1.57
salad	2	1.27
Falstaff	4	0.967
forest	12	0.489
battle	21	0.246
wit	34	0.037
fool	36	0.012
good	37	0
sweet	37	0

$\log_{10}(37/1)=1.57$

$\log_{10}(37/2)=1.27$

$\log_{10}(37/36)=0.012$

$\log_{10}(37/37)=0$

- A **tf-idf** weighted term-document matrix for four words in four Shakespeare plays.

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.246	0	0.454	0.520
good	0	0	0	0
fool	0.030	0.033	0.0012	0.0019
wit	0.085	0.081	0.048	0.054

TF-IDF Vector Models

- A **tf-idf vector model of a target word** is a vector with dimensions corresponding to all the words in the vocabulary.
  - The values in each dimension are the frequency with which the target word co-occurs with each neighboring context word, weighted by tf-idf.
  - *The model computes the similarity between two words x and y by taking the cosine of their tf-idf vectors; high cosine, high similarity*.

- To compare two documents:
  - Take the centroid of vectors of all the words in the document
  - Given k word vectors $w_1,\ldots,w_k$, **centroid document vector d** is:

$$d = \frac{w_1 + w_2 + \ldots + w_k}{k}$$

  - Given two documents, we can then compute their centroid document vectors d_1 and d_2, and estimate the similarity between the two documents by $\cos(d_1,d_2)$.

Pointwise Mutual Information (PMI)

- An alternative weighting function to tf-idf is called **PPMI (positive pointwise mutual information**).
  - PPMI draws on the intuition that best way to weigh the association between two words is to ask how much more the two words co-occur in our corpus than we would have a priori expected them to appear by chance.
- **Pointwise mutual information** is ***a measure of how often two events x and y occur, compared with what we would expect if they were independent***:

$$I(x,y) = \log_2 \frac{P(x,y)}{P(x)P(y)}$$

- **The pointwise mutual information between a target word w and a context word c** is then defined as:

$$\text{PMI}(w,c) = \log_2 \frac{P(w,c)}{P(w)P(c)}$$

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
- So, we just replace negative PMI values by 0

- **Positive PMI (PPMI) between a target word w and a context word c:**

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

Computing PPMI on a Term-Context Matrix

- We have a co-occurrence matrix with **W** rows (target words) and **C** columns (contexts), where f_{ij} gives the number of times word w_i occurs in context c_j.

	computer	data	result	pie	sugar	count(w)
cherry	2	8	9	442	25	486
strawberry	0	0	1	60	19	80
digital	1670	1683	85	5	4	3447
information	3325	3982	378	5	13	7703
count(context)	4997	5673	473	512	61	11716

- We can compute a PPMI matrix where PPMI_{ij} gives the PPMI value of word w_i with context c_j as follows:

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

$$\text{PPMI}_{ij} = \max\left(\log_2 \frac{p_{ij}}{p_{i*}p_{*j}}, 0\right)$$

Computing PPMI on a Term-Context Matrix – Example

- Compute PPMI(w=information, c=data),

$$P(\text{w=information, c=data}) = \frac{3982}{11716} = .3399$$

$$P(\text{w=information}) = \frac{7703}{11716} = .6575$$

$$P(\text{c=data}) = \frac{5673}{11716} = .4842$$

$$\text{PPMI(information,data)} = \log_2(.3399/(.6575 * .4842)) = .0944$$

	computer	data	result	pie	sugar	count(w)
cherry	2	8	9	442	25	486
strawberry	0	0	1	60	19	80
digital	1670	1683	85	5	4	3447
information	3325	3982	378	5	13	7703
count(context)	4997	5673	473	512	61	11716

joint probabilities computed from the counts

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

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

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

	p(w,context)					p(w)
	computer	data	result	pie	sugar	p(w)
cherry	0.0002	0.0007	0.0008	0.0377	0.0021	0.0415
strawberry	0.0000	0.0000	0.0001	0.0051	0.0016	0.0068
digital	0.1425	0.1436	0.0073	0.0004	0.0003	0.2942
information	0.2838	0.3399	0.0323	0.0004	0.0011	0.6575
p(context)	0.4265	0.4842	0.0404	0.0437	0.0052	

Computing PPMI on a Term-Context Matrix – Example

	p(w,context)					p(w)
	computer	data	result	pie	sugar	p(w)
cherry	0.0002	0.0007	0.0008	0.0377	0.0021	0.0415
strawberry	0.0000	0.0000	0.0001	0.0051	0.0016	0.0068
digital	0.1425	0.1436	0.0073	0.0004	0.0003	0.2942
information	0.2838	0.3399	0.0323	0.0004	0.0011	0.6575
p(context)	0.4265	0.4842	0.0404	0.0437	0.0052	

- **PPMI Matrix** $\text{PPMI}(\text{information},\text{data}) = \log_2(.3399/(.6575 * .4842)) = .0944$
  - The PPMI matrix showing the association between words and context words
  - Note that most of the 0 PPMI values are ones that had a negative PMI, we replace negative values by zero.

	computer	data	result	pie	sugar
cherry	0	0	0	4.38	3.30
strawberry	0	0	0	4.10	5.51
digital	0.18	0.01	0	0	0
information	0.02	0.09	0.28	0	0

Weighting PMI

Giving rare context words slightly higher probability

- PMI has the problem of being biased toward infrequent events;
  - very rare words tend to have very high PMI values.
- One way to reduce this bias is to slightly change computation for P(c), using a different function $P_\alpha(c)$ that raises contexts to power of α :

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

- Raise the context probabilities to $\alpha = 0.75$:
- This helps because $P_\alpha(c) > P(c)$ for rare c
- Consider two events, P(a) = .99 and P(b) = .01

$$P_\alpha(a) = \frac{.99^{.75}}{.99^{.75}+.01^{.75}} = .97 \qquad P_\alpha(b) = \frac{.01^{.75}}{.01^{.75}+.01^{.75}} = .03$$

Vector Semantics

- **Word2vec**

TF-IDF and PPMI Vectors

- **TF-IDF and PPMI vectors are**
  - **long vectors** with dimensions corresponding to the words in the vocabulary, i.e. their lengths are |V| which is the number of words in the language and |V| can be around 50,000.
  - **sparse**, i.e ***most of the elements in those vectors are zero.***
    - There are no negative values.

- In most of NLP tasks, **dense vectors** work better than **sparse vectors**.

- **Dense vectors** are **short**, and their lengths are between 50 and 1000.
  - They are **dense**, i.e. ***their most-elements are non-zero.***
  - The values will be *real-valued numbers that can be negative*.

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 they are in distinct dimensions
  - a neighboring word with ***car*** and a neighboring word with **automobile** should be similar, but they are not similar.

- **In practice, dense vectors work better.**

Word2vec

- One of the most popular embedding method is **Word2vec** (Mikolov et al. 2013).
- **Word2vec** embeddings are **static embeddings**, meaning that the method learns one fixed embedding for each word in the vocabulary.
- Later we will talk **dynamic contextual embeddings** methods like BERT, ELMo representations, in which *the vector for each word is different in different contexts*.
- The **Word2vec methods** are very fast, efficient to train and easily available online with code and pretrained embeddings.
  - **Main dense vector idea**: **predict** rather than **count .**
- Popular dense **static embeddings**:
  - **Word2vec** (Mikolov et al.) https://code.google.com/archive/p/word2vec/
  - **Fasttext** http://www.fasttext.cc/
  - **Glove** (Manning et al.) http://nlp.stanford.edu/projects/glove/

Word2vec

- The **intuition of word2vec** is:
  - 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 the word embeddings**.

Brilliant insight: ***Use running text (plain 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.

Word2vec: Skip-Gram Algorithm
skip-gram with negative sampling (SGNS)

The ***intuition of skip-gram*** is:

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**

Skip-Gram Training Data

- Assume that **context words** are those in +/- 2 word window.
- A training sentence:

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

c1 c2 **target** c3 c4

Skip-Gram Goal

- Our goal is to train a classifier such that,
  - Given a **tuple (w,c)** of **a target word t paired with a context word c**
    - (apricot, jam)
    - (apricot, aardvark)
  - it will return the **probability that c is a real context word** (**true** for jam, **false** for aardvark):

P(+ | w,c) Probability that word c is a real context word for w

P(− | w,c) = 1−P(+|w,c) Probability that word c is not a real context word for w

How to Compute P(+|w,c)?

The **intuition of the skipgram model** is to base this probability on similarity:

- ***A word is likely to occur near the target*** if ***its embedding is similar to the target embedding***.

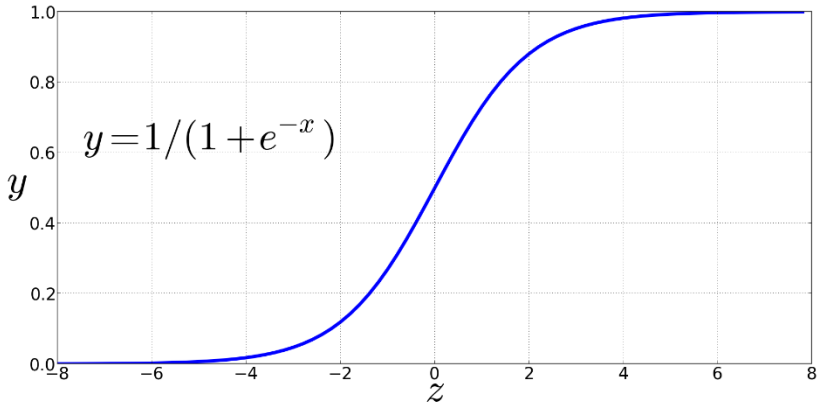
- ***Two vectors are similar if they have a high dot product.***
  - **Similarity(w,c)** $\approx$ $\mathbf{w} \cdot \mathbf{c}$
  - The dot product **w.c** is not a probability, it's just a number ranging from $-\infty$ to $+\infty$.

How to Compute P(+|t,c)?

Turning dot product into a probability

- To turn **dot product** into a probability, we'll use **logistic** or **sigmoid function** σ(x).

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



- **The probability that word c is a real context word for target word t is:**

$$P(+|w,c) = \sigma(c \cdot w) = \frac{1}{1+\exp(-c \cdot w)}$$

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

$$= \sigma(-c \cdot w) = \frac{1}{1+\exp(c \cdot w)}$$

How to Compute P(+|t,c)?

For all the context words:

- We need to take account of the multiple context words in the window.
- Skip-gram makes the strong but very useful simplifying assumption that all context words are independent, allowing us to just multiply their probabilities:

$$P(+|w, c_{1:L}) = \prod_{i=1}^{L} \sigma(c_i \cdot w)$$

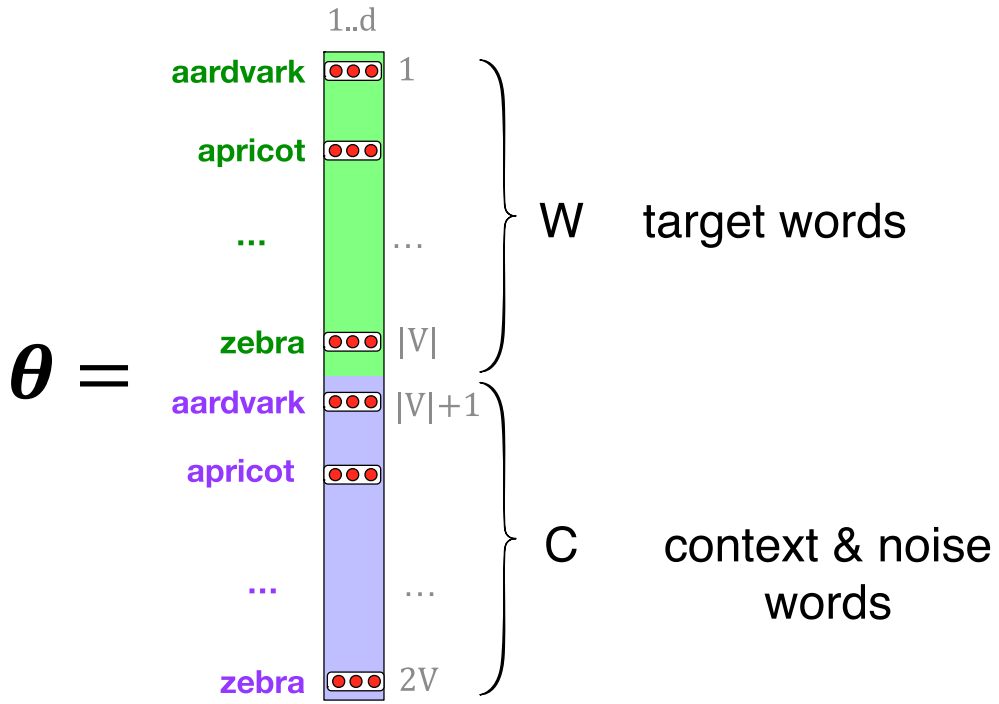
$$\log P(+|w, c_{1:L}) = \sum_{i=1}^{L} \log \sigma(c_i \cdot w)$$

  - Skip-gram trains a probabilistic classifier that, given a test target word t and its context window of k words $c_{1:L}$, assigns a probability based on how similar this context window is to the target word.
  - The probability is based on applying the logistic (sigmoid) function to the dot product of the embedings of the target word with each context word.

Learning skip-gram embeddings

skip-gram embeddings to be learned

- **Skip-gram** actually stores two embeddings for each word, one for the word as a target, and one for the word considered as context.
- Thus the parameters we need to learn are two matrices **W** and **C**, each containing an embedding for every one of the |V| words in the vocabulary



Learning skip-gram embeddings

Skip-Gram Training Data

- **Training sentence:**

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

c1 c2 **target w** c3 c4

- **Training data:** pairs centering on **apricot**
- Assume a +/- 2 word window is used.

Learning skip-gram embeddings

Skip-Gram Training

- **Training sentence:**

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

c1 c2 **w** c3 c4

positive examples +

w	**c**
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

- For each positive example, we'll create ***k* negative examples**.
- Using ***noise*** words
  - Noise word is any random word that isn't *target word w (apricot)*

Learning skip-gram embeddings

Skip-Gram Training

- **Training sentence:**

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

c1 c2 **w** c3 c4

- Create k (=2) negative examples.
- We'll have 2 negative examples for each positive example w,cpos.
- The noise words are chosen according to their weighted unigram frequency.

positive examples +		**negative examples -**			
w	**c**	**w**	**c**	**w**	**c**
apricot	tablespoon	apricot	aardvark	apricot	twelve
apricot	of	apricot	puddle	apricot	hello
apricot	jam	apricot	where	apricot	dear
apricot	a	apricot	coaxial	apricot	forever

Learning skip-gram embeddings

Choosing noise words

- Could pick w as a noise word according to their unigram frequency P(w)
- More common to chosen then according to $P_\alpha(w)$

$$P_\alpha(w) = \frac{count(w)^\alpha}{\sum_w count(w)^\alpha}$$

- $\alpha = 0.75$ works well because it gives rare noise words slightly higher probability
- To show this, imagine two events P(a)=.99 and P(b) = .01:

$$P_\alpha(a) = \frac{.99^{.75}}{.99^{.75} + .01^{.75}} = .97$$

$$P_\alpha(b) = \frac{.01^{.75}}{.99^{.75} + .01^{.75}} = .03$$

Learning skip-gram embeddings

Setup

- Let's represent words as vectors of some length (say 300), randomly initialized.
- So, we start with 300*V random parameters
- Over the entire training set, we'd like to **adjust those word vectors such that we**
  - **Maximize the similarity of the target word w, context word c pairs (w,c) drawn from the positive data**
  - **Minimize the similarity of the pairs (w,c) drawn from the negative data.**

- We'll start with 0 or random weights
- Then **adjust the word weights to make the positive pairs more likely and the negative pairs less likely** over the entire training set:

Learning skip-gram embeddings

Objective Criteria

- We want to maximize

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

- **Maximize the + label for the pairs from the positive training data, and the – label for the pairs sample from the negative data**.

Learning skip-gram embedings

Loss function for one target word w with c_{pos} , c_{neg1} ...c_{negk}

- Focusing in on one word/context pair (w,c) with its k noise words $\text{neg}_1 \ldots \text{neg}_k$, the **loss function** is:

$$
\begin{aligned}
L_{CE} &= -\log\left[P(+|w,c_{pos})\prod_{i=1}^{k}P(-|w,c_{neg_i})\right] \\
&= -\left[\log P(+|w,c_{pos}) + \sum_{i=1}^{k}\log P(-|w,c_{neg_i})\right] \\
&= -\left[\log P(+|w,c_{pos}) + \sum_{i=1}^{k}\log\left(1-P(+|w,c_{neg_i})\right)\right] \\
&= -\left[\log \sigma(c_{pos}\cdot w) + \sum_{i=1}^{k}\log\sigma(-c_{neg_i}\cdot w)\right]
\end{aligned}
$$

- That is, **we want to maximize the dot product of the word with the actual context word, and minimize the dot products of the word with the k negative sampled non-neighbor words.**
  - all multiplied because we assume independence.

Learning skip-gram embeddings

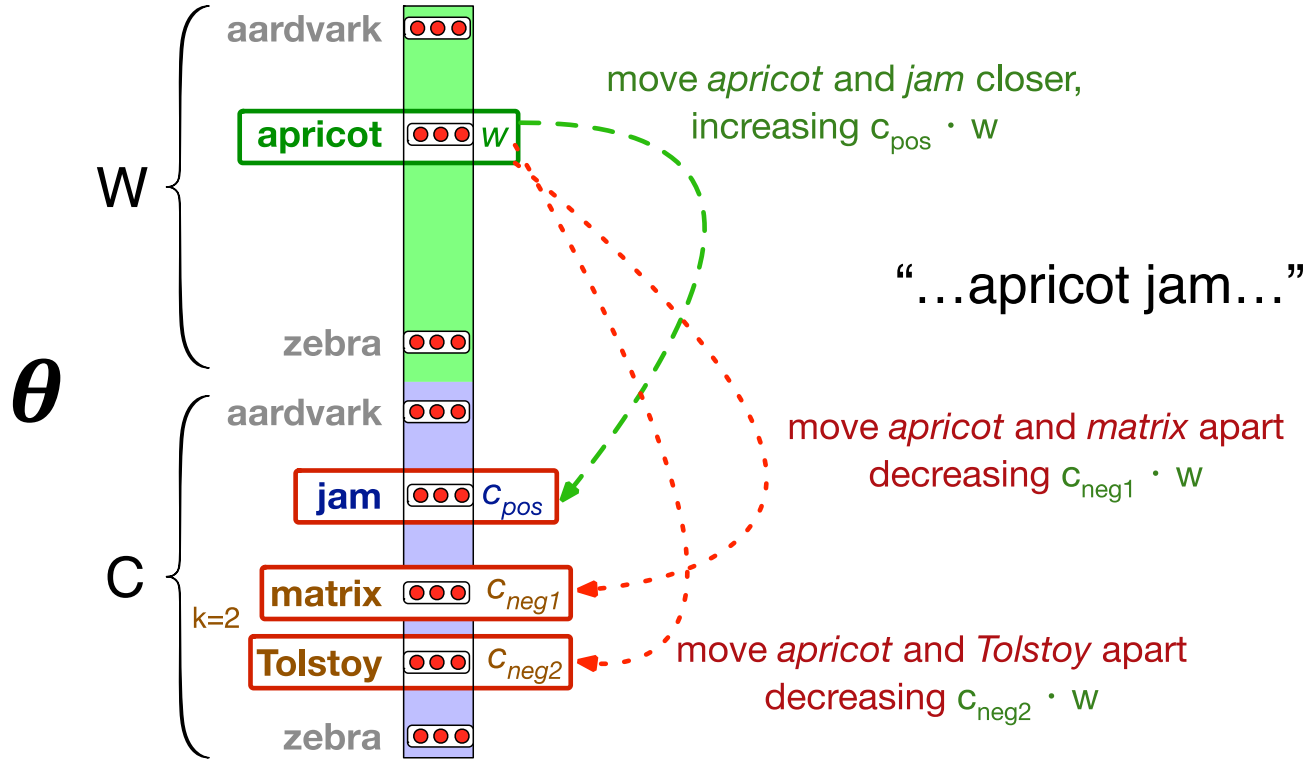
Train using gradient descent

- We can then use **stochastic gradient descent to train to this objective**,
  - iteratively modifying the parameters (the embeddings for each target word t and each context word or noise word c in the vocabulary) to maximize the objective.
- **Skip-gram model** actually learns **two separate embeddings for each word w**:
  - **target embedding w** and
  - **context embedding c.**
- These embeddings are stored in **two matrices,**
  - **target matrix W and**
  - **context matrix C**.
- We'll adjust the word weights to make the positive pairs more likely and the negative pairs less likely, over the entire training set.

Learning skip-gram embeddings

Intuition of one step of gradient descent

- The skip-gram model tries to shift embeddings so the target embeddings (here for apricot) are closer to (have a higher dot product with) context embeddings for nearby words (here jam) and further from (lower dot product with) context embeddings for noise words that don't occur nearby (here Tolstoy and matrix).



Learning skip-gram embeddings

The derivatives of the loss function

- To get the **gradient**, we need to take the derivative of the loss function with respect to the different embeddings.
  - We'll need a derivate for the C embeddings and for the W embeddings.
  - And we'll want a different derivative for the weights for the positive context examples and the negative context examples, since we want to move them in opposite directions.

- The derivatives are

$$L_{CE} = -\left[\log \sigma(c_{pos} \cdot w) + \sum_{i=1}^{k} \log \sigma(-c_{neg_i} \cdot w)\right]$$

$$\frac{\partial L_{CE}}{\partial c_{pos}} = [\sigma(c_{pos} \cdot w) - 1]w$$

$$\frac{\partial L_{CE}}{\partial c_{neg}} = [\sigma(c_{neg} \cdot w)]w$$

$$\frac{\partial L_{CE}}{\partial w} = [\sigma(c_{pos} \cdot w) - 1]c_{pos} + \sum_{i=1}^{k} [\sigma(c_{neg_i} \cdot w)]c_{neg_i}$$

Learning skip-gram embeddings

Update equation in stochastic gradient descent

- Just as in logistic regression, then, the learning algorithm starts with randomly initialized W and C matrices, and then walks through the training corpus using gradient descent to move W and C so as to maximize the objective.

- **Update equations going from time step t to t + 1 in stochastic gradient descent**.

$$
\begin{aligned}
c_{pos}^{t+1} &= c_{pos}^{t} - \eta[\sigma(c_{pos}^{t} \cdot w^{t}) - 1]w^{t} \\
c_{neg}^{t+1} &= c_{neg}^{t} - \eta[\sigma(c_{neg}^{t} \cdot w^{t})]w^{t} \\
w^{t+1} &= w^{t} - \eta \left[[\sigma(c_{pos} \cdot w^{t}) - 1]c_{pos} + \sum_{i=1}^{k} [\sigma(c_{\mathrm{neg}_i} \cdot w^{t})]c_{neg_i} \right]
\end{aligned}
$$

Semantic properties of embeddings

Semantic properties of embeddings

Similarity depends on window size C

- One parameter that is relevant to both sparse tf-idf vectors and dense word2vec vectors is the **size of the context window** used to collect counts.

  Short context windows ➔most similar words to a target word w tend to be semantically similar words with same parts of speech.

  Long context windows ➔ the highest cosine words to a target word w tend to be words that are topically related but not similar.

  Example:

  window of +/-2 ➔ the most similar words to the word **Hogwarts** (from the Harry Potter series) were names of other fictional schools: **Sunnydale** (from Buffy the Vampire Slayer) or **Evernight** (from a vampire series).

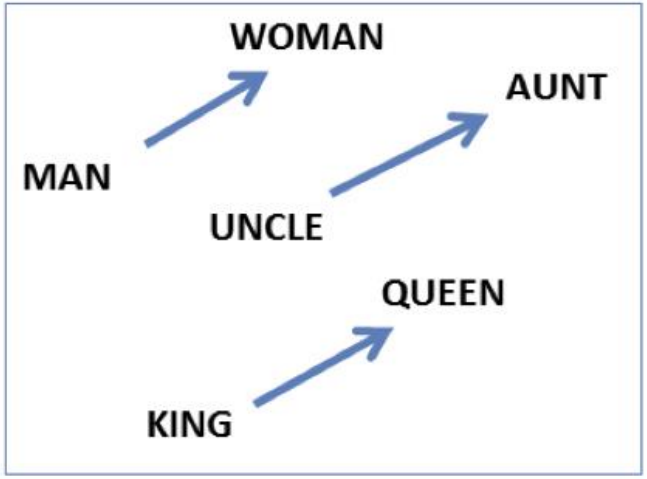
  window of +/-5 ➔ the most similar words to **Hogwarts** were other words topically related to the Harry Potter series: **Dumbledore**, **Malfoy**, and **half-blood**.

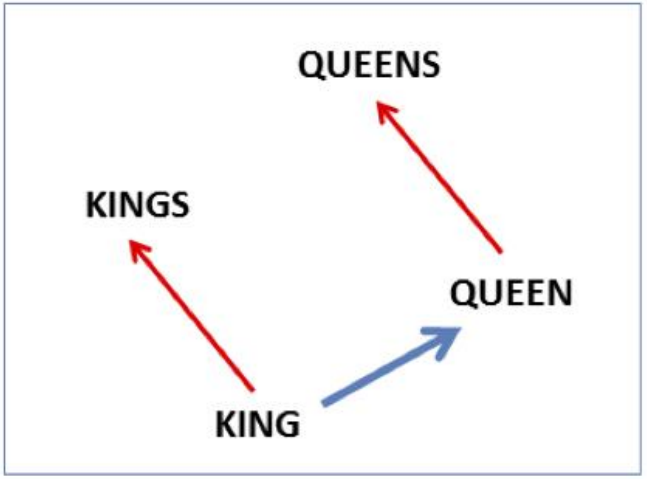
Analogy: Embeddings capture relational meaning!

- Another semantic property of embeddings is their **ability to capture relational meanings.**

vector(*'king'*) - vector(*'man'*) + vector(*'woman'*) ≈ vector('queen')

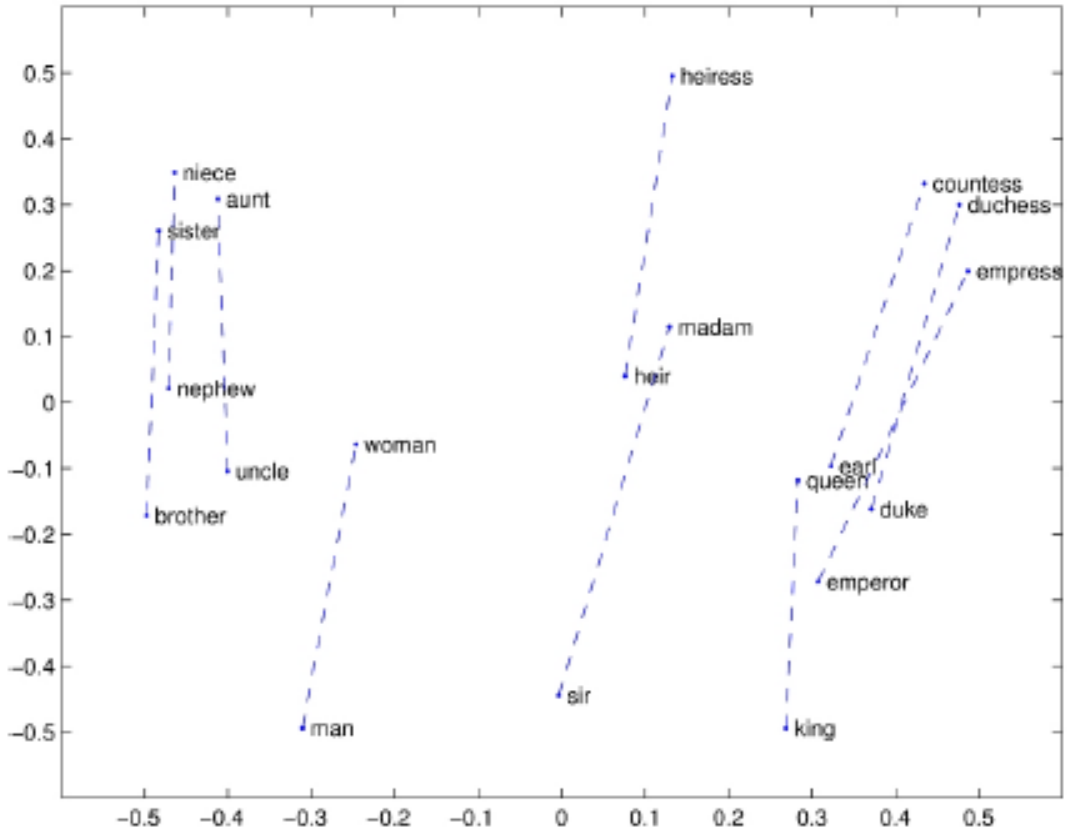
vector(*'Paris'*) - vector(*'France'*) + vector(*'Italy'*) ≈ vector('Rome')





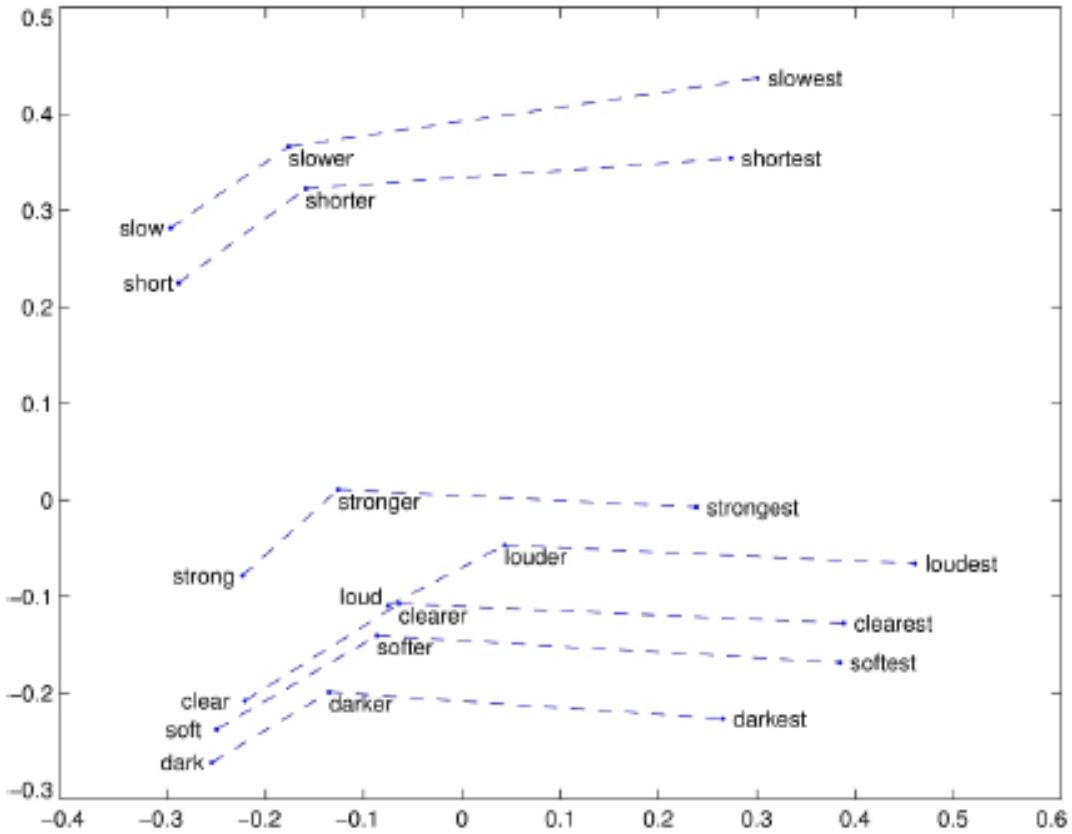
Analogy: Embeddings capture relational meaning!

- Relational properties of the vector space, shown by projecting vectors onto two dimensions



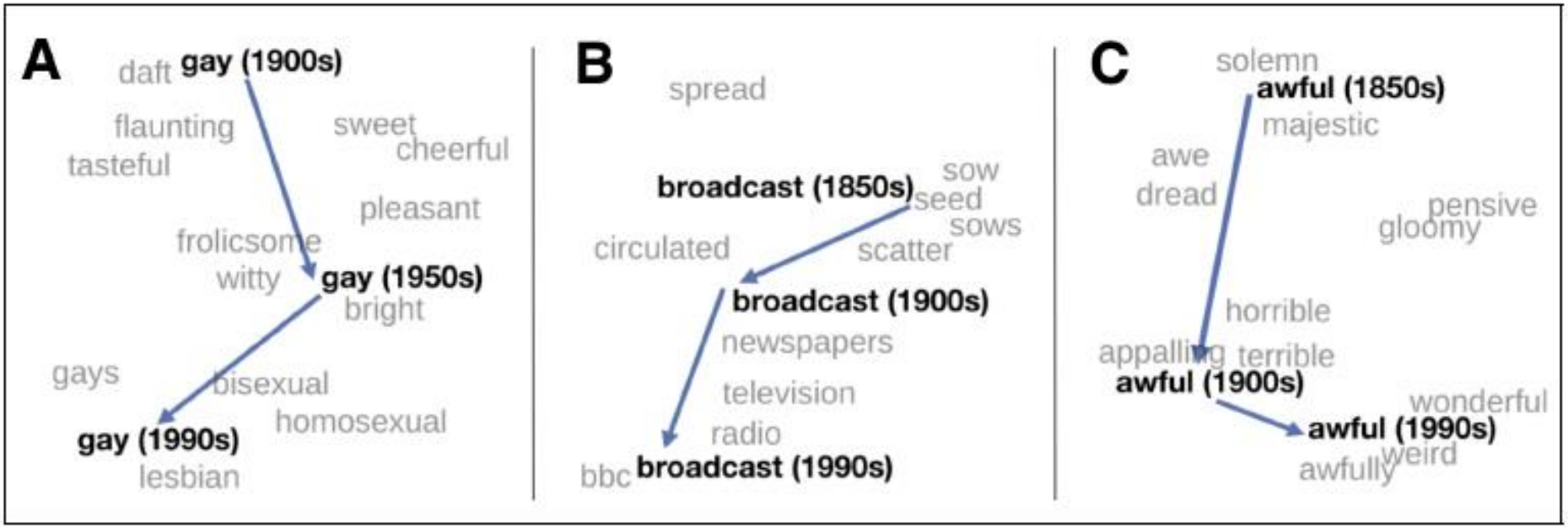
Analogy: Embeddings capture relational meaning!

- offsets seem to capture comparative and superlative morphology



Embeddings and Historical Semantics

- Train embeddings on old books to study changes in word meaning!!



Evaluating Vector Models

- 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*

Vector Semantics: Summary

- In **vector semantics**, **a word is modeled as a vector** — a point in high-dimensional space, also called an **embedding**.
- **Vector semantic models fall into two classes**: **sparse** and **dense**.

Sparse Models:

- In sparse models like tf-idf **each dimension corresponds to a word** in the vocabulary.
- Cell in sparse models are **functions of co-occurrence counts**.
- The **term document matrix** has rows for each word (**term**) in the vocabulary and a column for each document.
- The **word-context matrix** has a row for each (target) word in the vocabulary and a column for each context term in the vocabulary.
- A **common sparse weighting is tf-idf**, which weights each cell by its **term frequency** and **inverse document frequency**.
- **Word and document similarity** is computed by computing the **dot product between vectors**. The cosine of two vectors—a normalized dot product—is the most popular such metric.
- **PPMI (pointwise positive mutual information)** is an alternative weighting scheme to tf-idf.

Vector Semantics: Summary

Dense Models:

- **Dense vector models have dimensionality 50-300** and the dimensions are harder to interpret.
- The **word2vec** family of models, including **skip-gram**, is a **popular efficient way to compute dense embeddings**.
- **Skip-gram** trains a **logistic regression classifier** to compute the probability that two words are 'likely to occur nearby in text'.
  - This probability is computed from the dot product between the embeddings for the two words,
- Skip-gram uses **stochastic gradient descent to train the classifier**, by learning embeddings that have a high dot-product with embeddings of words that occur nearby and a low dot-product with noise words.
- Other important embedding algorithms include **GloVe**, and **fasttex**t.