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.***

- 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.***

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 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).
- 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 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

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

- Asssume 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 (t,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(+ | t,c)	Probability that word c is nota real context word for t
P(− | t,c) = 1−P(+|t,c)	Probability that word c is **not** a real context word for t

How to Compute P(+|t,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***.

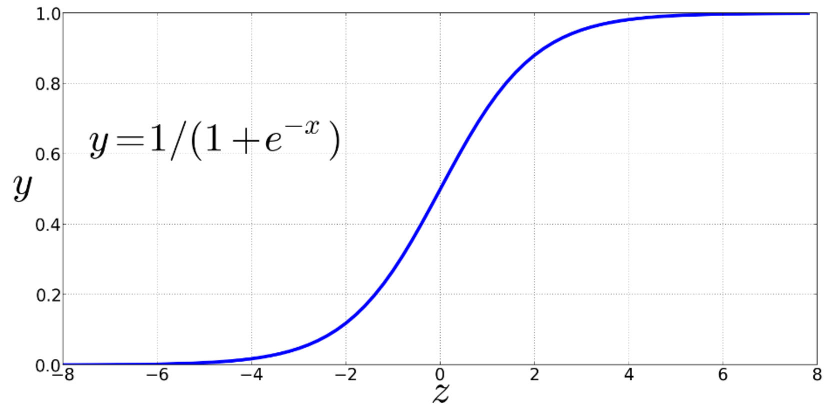
- ***Two vectors are similar if they have a high dot product***.
  - **Similarity(t,c) $\propto$ t · c**
  - The dot product **t.c** is not a probability, it's just a number ranging from 0 to ∞.

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** $\sigma(x)$.

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



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

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

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

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:

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

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

  – Skip-gram trains a probabilistic classifier that, given a test target word t and its context window of k words $c_{1:k}$, 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 embeddings of the target word with each context word.

Learning skip-gram embeddings

Skip-Gram Training Data

- **Training sentence:**

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

  c1 c2 **t** c3 c4

- **Training data:** pairs centering on **apricot**
- Asssume 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 **t** c3 c4

positive examples +

t	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 t (apricot)*

Learning skip-gram embeddings

Skip-Gram Training

- **Training sentence:**

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

c1 c2 **t** c3 c4

Create k (=2) negative examples.

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

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:

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

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, context word pairs (t,c) drawn from the positive data**
  - **Minimize the similarity of the (t,c) pairs 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 embeddings
Objective Criteria

- Focusing in on one word/context pair (t,c) with its k noise words $n_1 \ldots n_k$, the learning objective L is:

$$
\begin{aligned}
L(\theta) &= \log P(+|t,c) + \sum_{i=1}^{k} \log P(-|t,n_i) \\
&= \log \sigma(c \cdot t) + \sum_{i=1}^{k} \log \sigma(-n_i \cdot t) \\
&= \log \frac{1}{1+e^{-c \cdot t}} + \sum_{i=1}^{k} \log \frac{1}{1+e^{n_i \cdot t}}
\end{aligned}
$$

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

Learning skip-gram embeddings

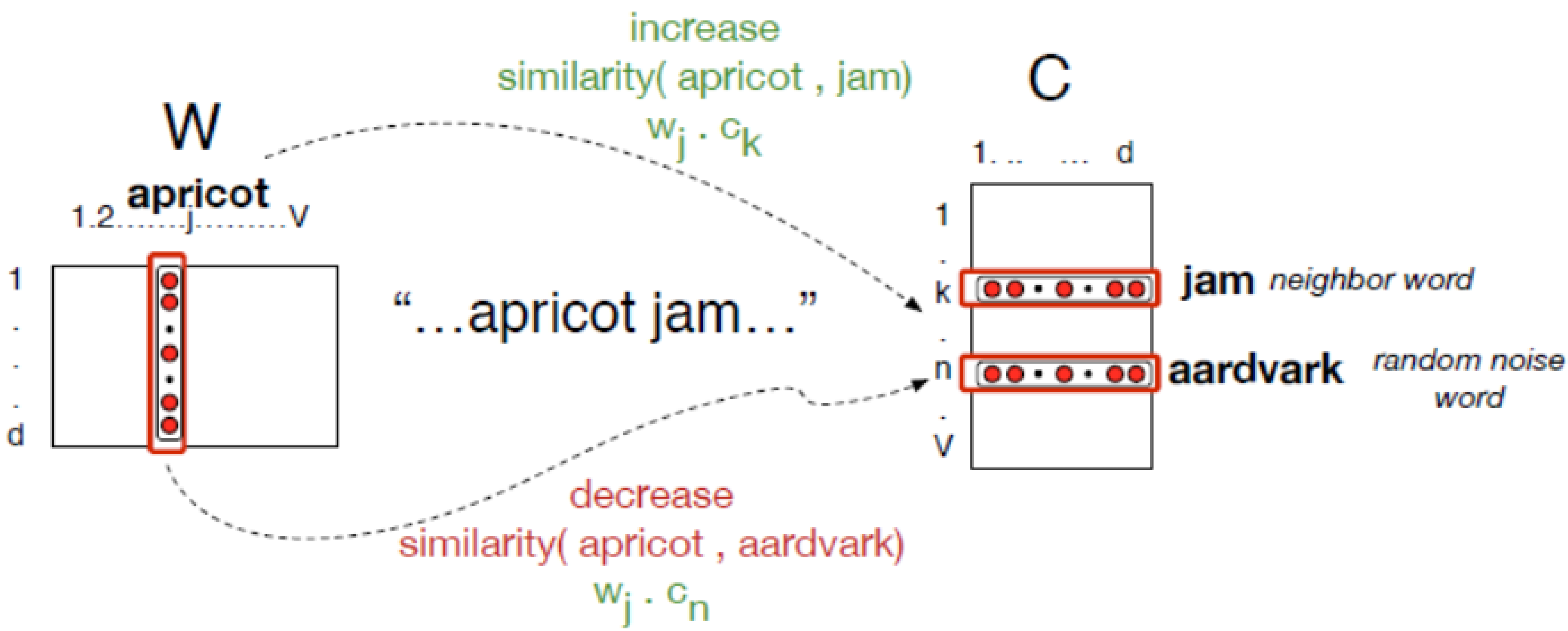
Train using gradient descent

- 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 t** and
  - **context embedding c.**
- These embeddings are stored in **two matrices,**
  - **target matrix W and**
  - **context matrix C**.

Learning skip-gram embeddings

Train using gradient descent

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



Learning skip-gram embeddings

Train using gradient descent

- Just as in logistic regression, the learning algorithm starts with randomly initialized W and C matrices, and then walks through the training corpus using gradient descent to update W and C so as to maximize the objective criteria.
- Thus matrices W and C function as the parameters θ that logistic regression is tuning.
- Once embeddings are learned, we'll have two embeddings for each word w_i: t_i and c_i.
  - We can choose to throw away the C matrix and just keep W, in which case each word i will be represented by the vector t_i.
  - Alternatively we can add the two embeddings together, using the summed embedding $t_i + c_i$ as the new d-dimensional embedding

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*

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).

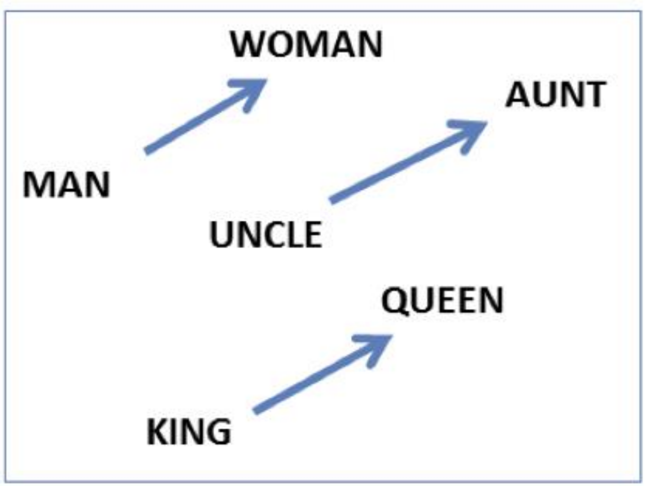
  window of +/-5 ➔ the most similar words to **Hogwarts** were other words topically related to the Harry Potter series: **Dumbledore**, **Malfoy**, and **half-blood**.

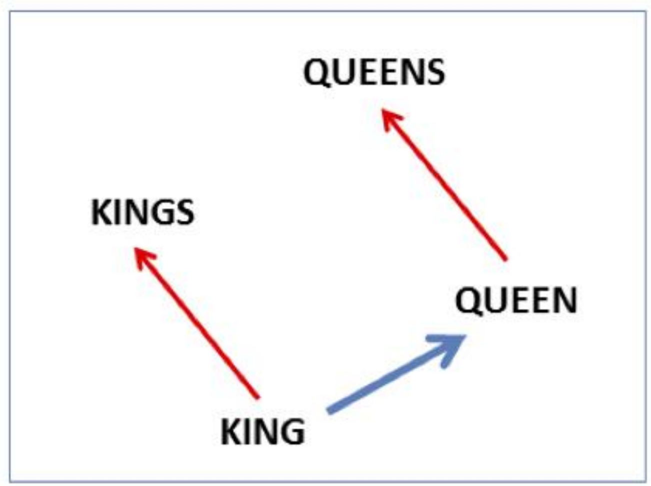
Analogy: Embeddings capture relational meaning!

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

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

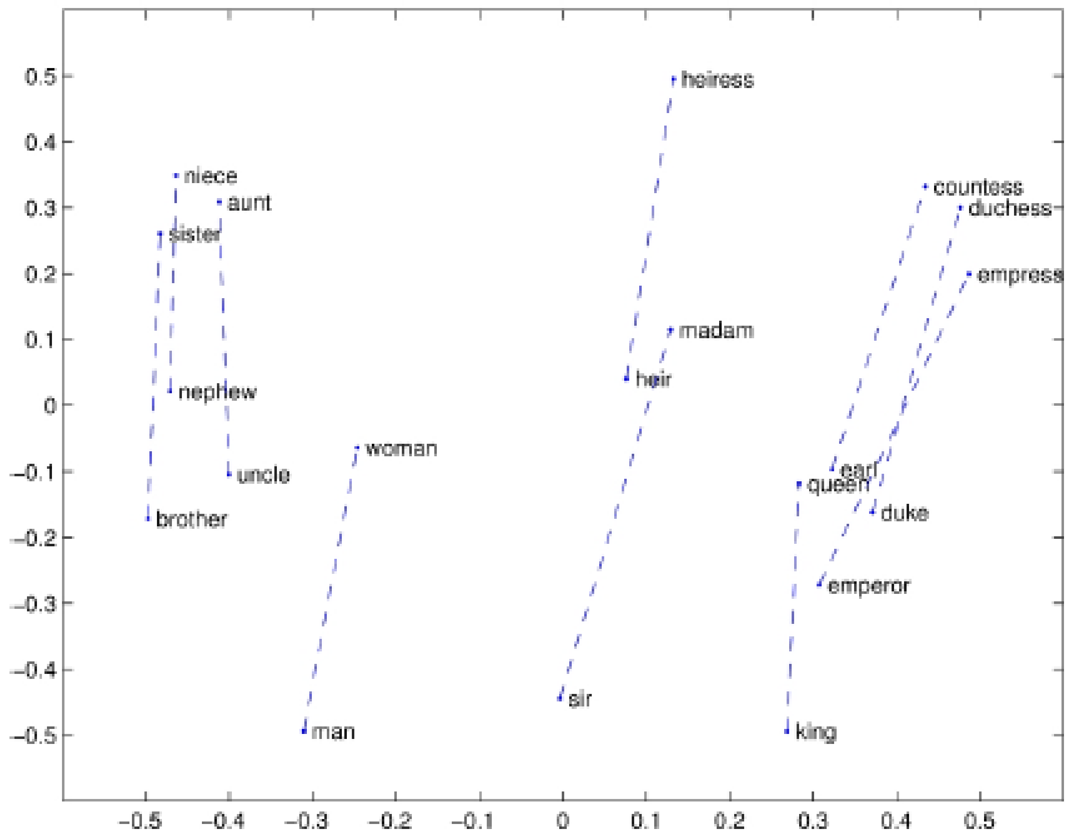
vector(*'Paris'*) - vector(*'France'*) + vector(*'Italy'*) ≈ vector('Rome')





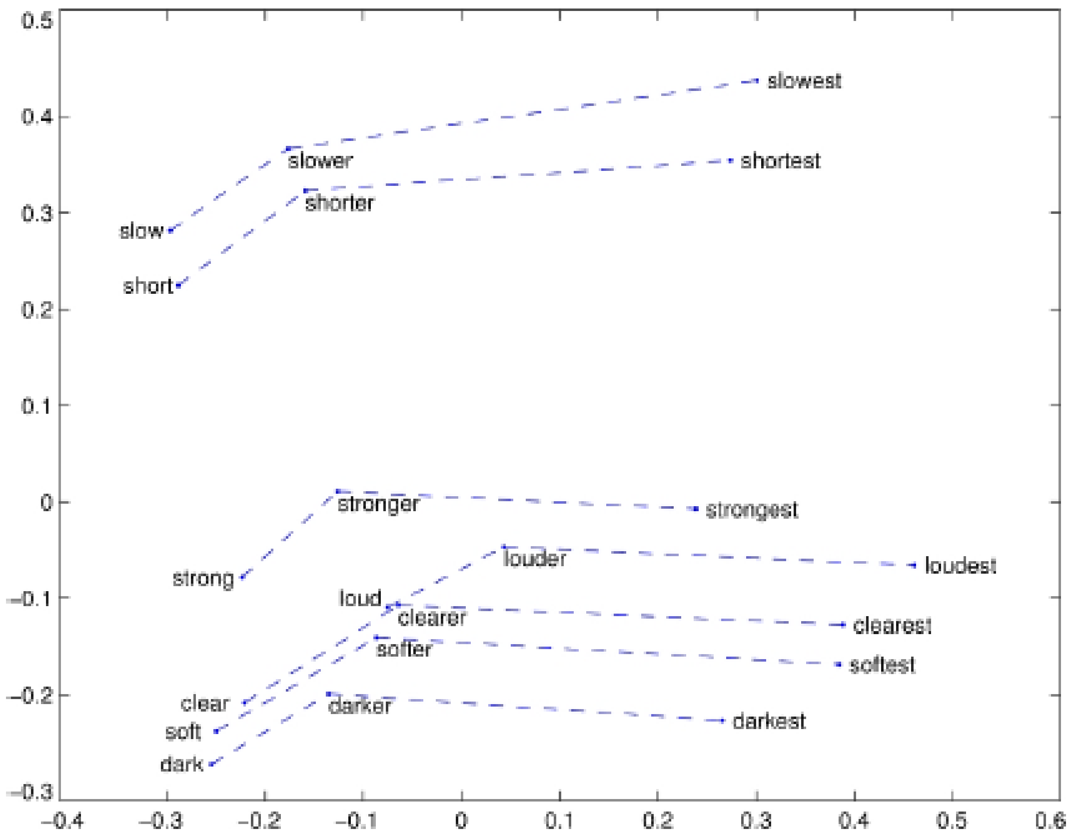
Analogy: Embeddings capture relational meaning!

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



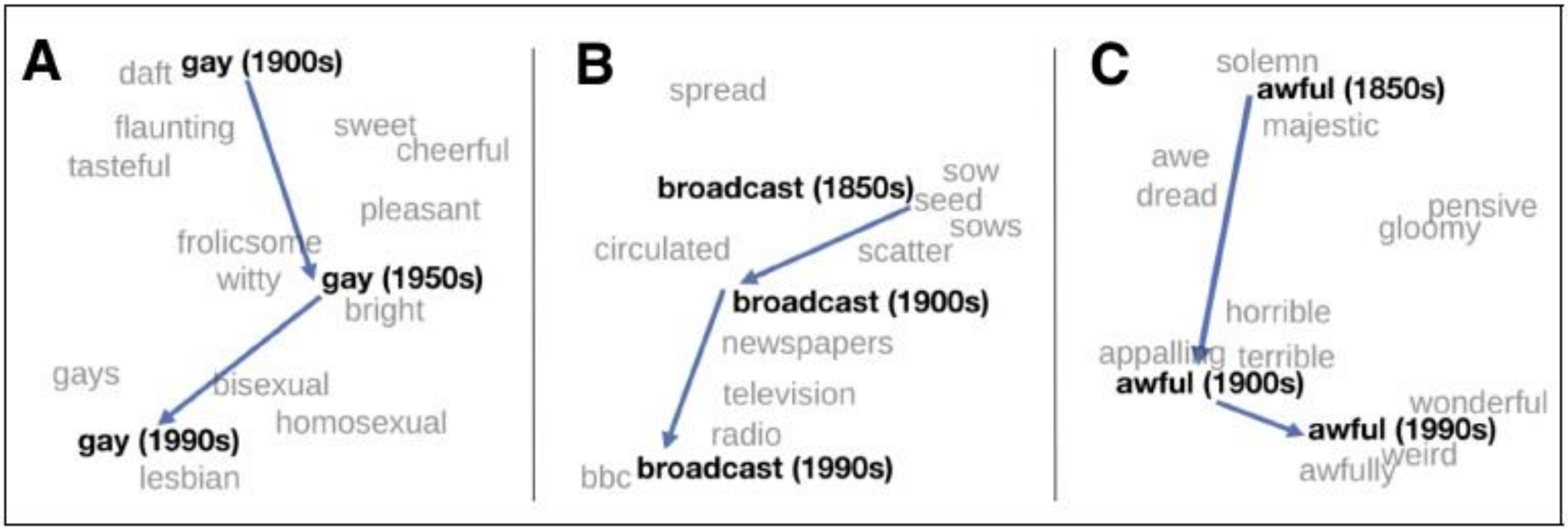
Analogy: Embeddings capture relational meaning!

- offsets seem to capture comparative and superlative morphology



Embeddings and Historical Semantics

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



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 ocur nearby and a low dot-product with noise words.
- Other important embedding algorithms include **GloVe**, and **fasttex**t.