PPOL 628: Text as Data – Computational Linguistics for Social Scientists

Class 12: Word Embeddings

Today

- Lecture: Word Embeddings.
  - Introduction to word embeddings.
  - Fitting Word2Vec.
  - Potential social science applications.

- Lab: word_embeddings.R

- Website: github.com/matthewjdenny/PPOL_628_Text_As_Data

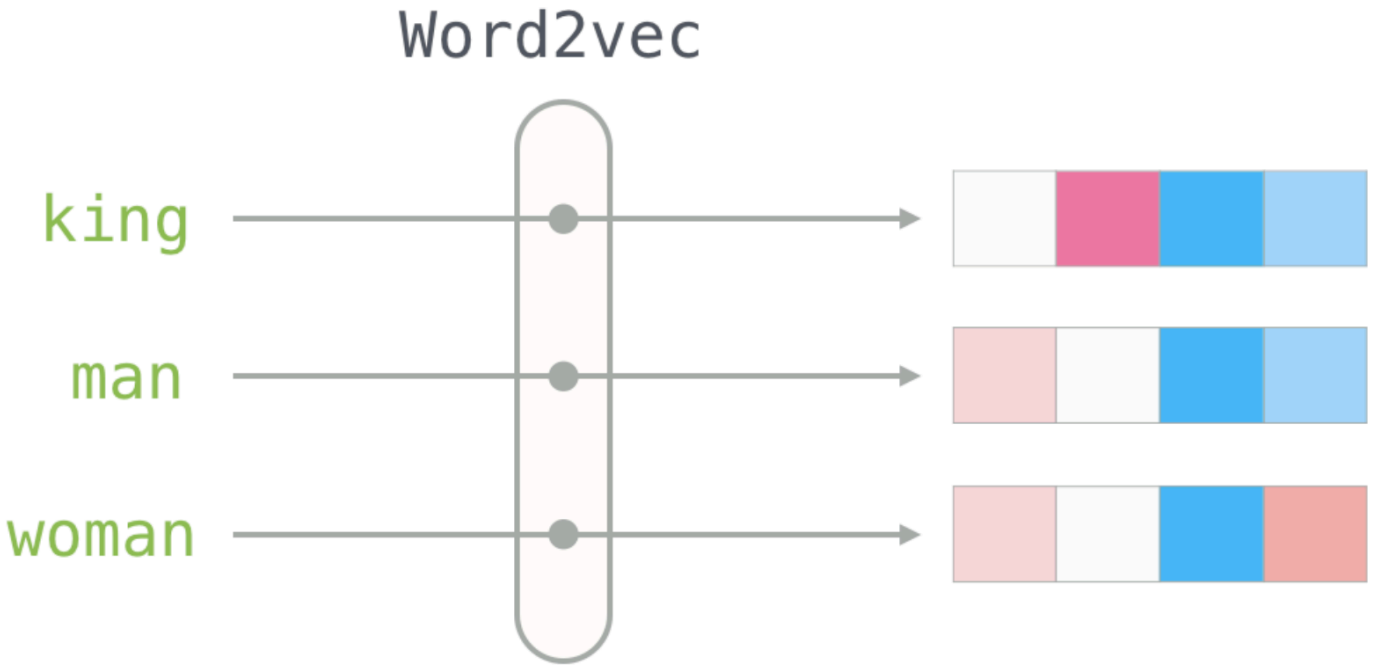
Word Embeddings

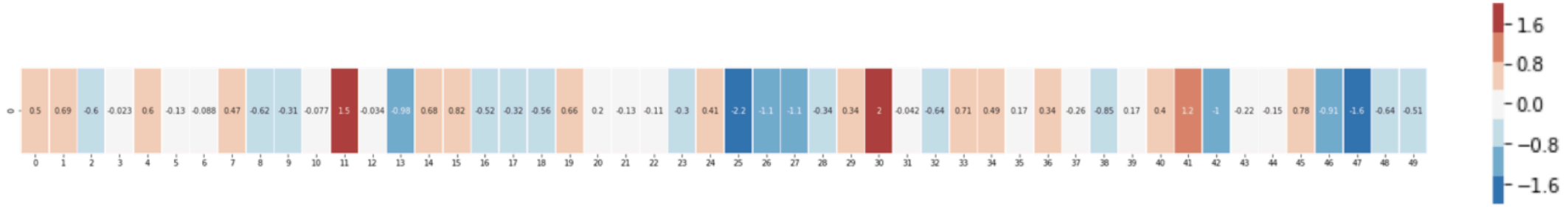
- Humans have learned understand words as connected to meanings, and are able to think of synonyms, make analogies, etc.
  - In order for computers to mimic this type of behavior, we have to represent words in a way that computers can "understand".

- One approach is to represent them as points in a (50-1000 dimensional) vector space. Meaning of words mapped to dimensions of vector space.
  - Turns out this approach works well in lots of NLP tasks – social science is still searching for a homerun application.

- Many papers + tutorials (which I screenshot from liberally in these slides):
  - Visualization: http://jalammar.github.io/illustrated-word2vec/
  - Jurafsky Slides: https://web.stanford.edu/~jurafsky/slp3/slides/vector2.pdf
  - Tutorial: http://u.cs.biu.ac.il/~gonenhi/slides/WiDS_tutorial.pdf
  - GloVe Website: https://nlp.stanford.edu/projects/glove/

Words Represented as Numerical Vectors





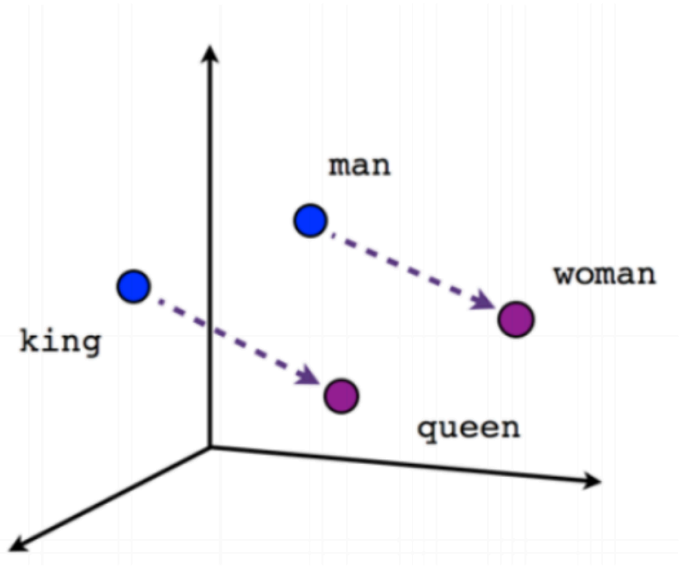
Word Embeddings

- There has been a huge amount of research into word embeddings or “vector space representations”, particularly as input for deep learning NLP models.
  - Thousands of academic papers.
  - Typically used as a form of preprocessing for deep learning algorithms.
  - Allows individual words to have some “intrinsic” meaning.

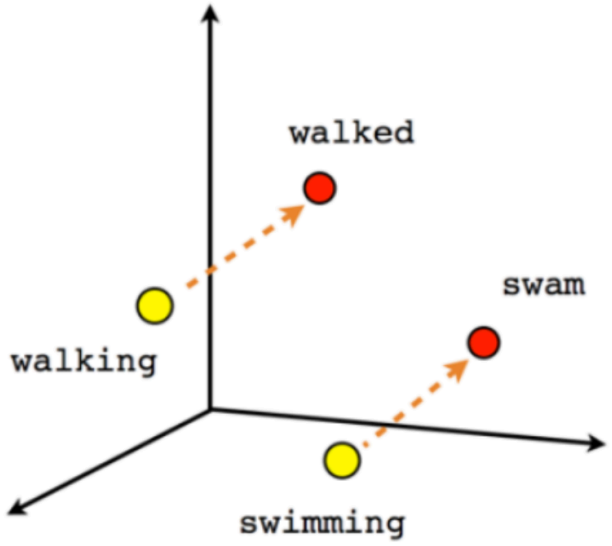
- Example: First ten dimensions of embedding for the word “football” in the GloVe 200 dimensional embedding space:
  - {-0.11, -0.34, -0.99, 0.85, 0.1, 0.74, -0.1, 0.36, 1.2, …}

Representing Relationships Between Words

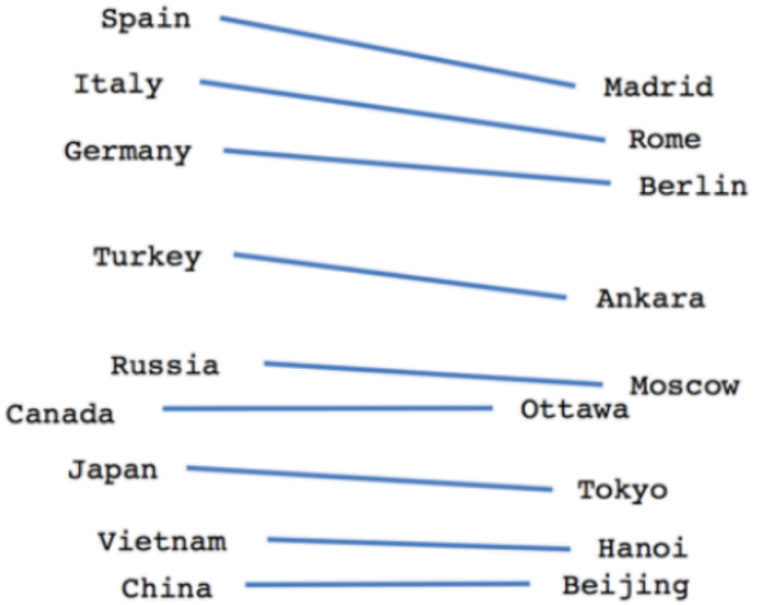
- Relative position of words in vector or embedding space can tell us about relationships among words.



Male-Female

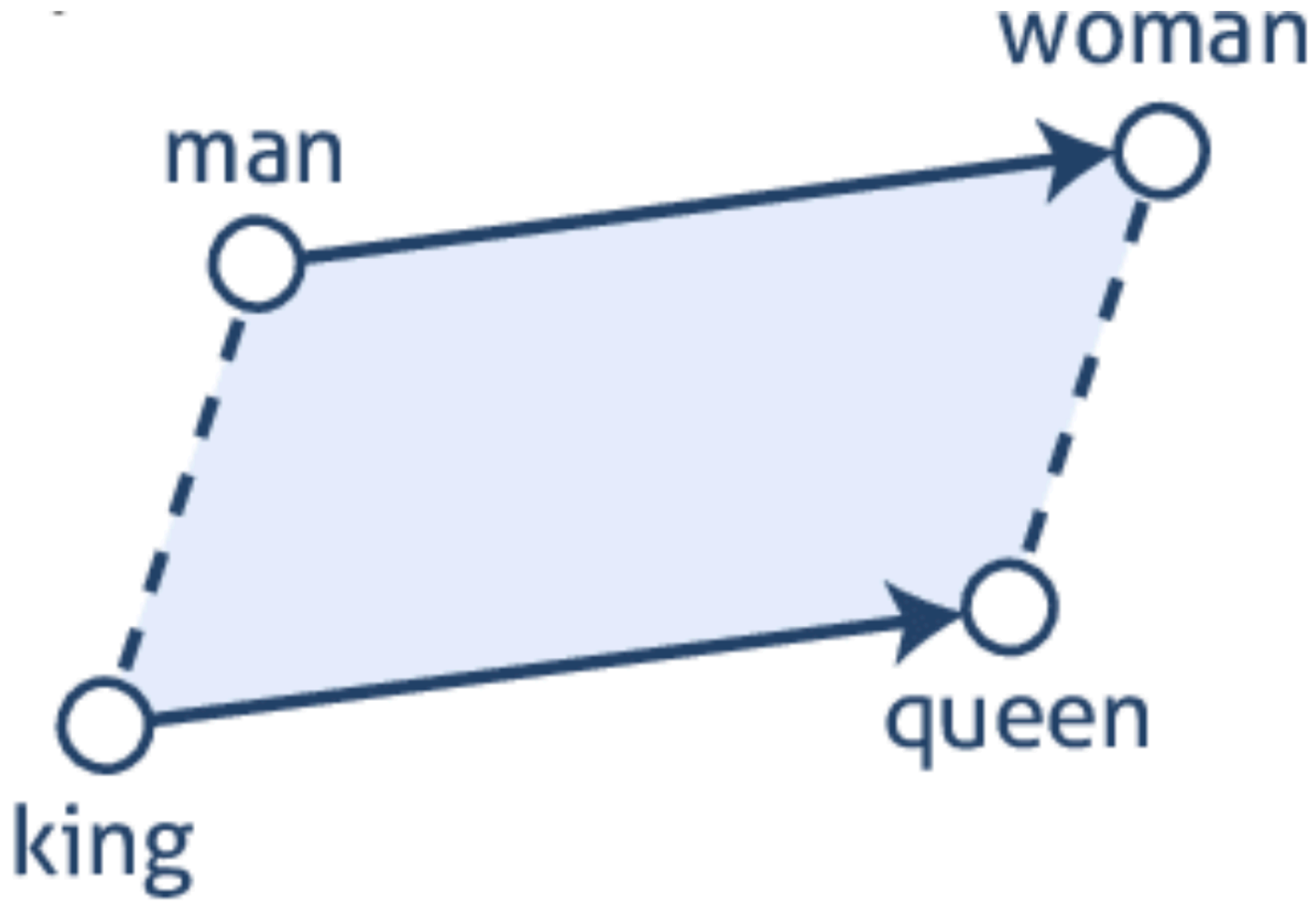


Verb tense



Country-Capital

Solving Analogies



Representation Learning

- Word embeddings are cool, and useful in a number of settings, but how do we generate them?
  - Answers come from field of “representation learning” where we use a machine learning model to learn the vector space representation of each word.

- Common models include Google’s Word2Vec, Stanford’s GloVe, Fasttext, etc. We will focus on the Word2Vec model in this lecture.
  - Goal of all of these models is to take a very large collection of documents as input and return a *k*-dimensional output numerical representation of each of the words in that corpus.
  - To have good performance, these models rely on absolutely enormous training corpora. E.g. Standard Word2Vec embeddings trained on **100 Billion** words of text.
  - Typically have a neural network model architecture.

Word2Vec

- Basic intuition: train a binary classifier to predict words that will show up close to each other. Use the weights the classifier learns for this task as the vector space embedding for each word.

- Magic advantage here is we do not need any human labeled training data -- we can just use the co-occurrence of terms near each other in real documents as our “training data”.

- Relatively fast to train – but is dependent on having lots of data.

- We will cover: "skip-gram with negative sampling" (SGNS)

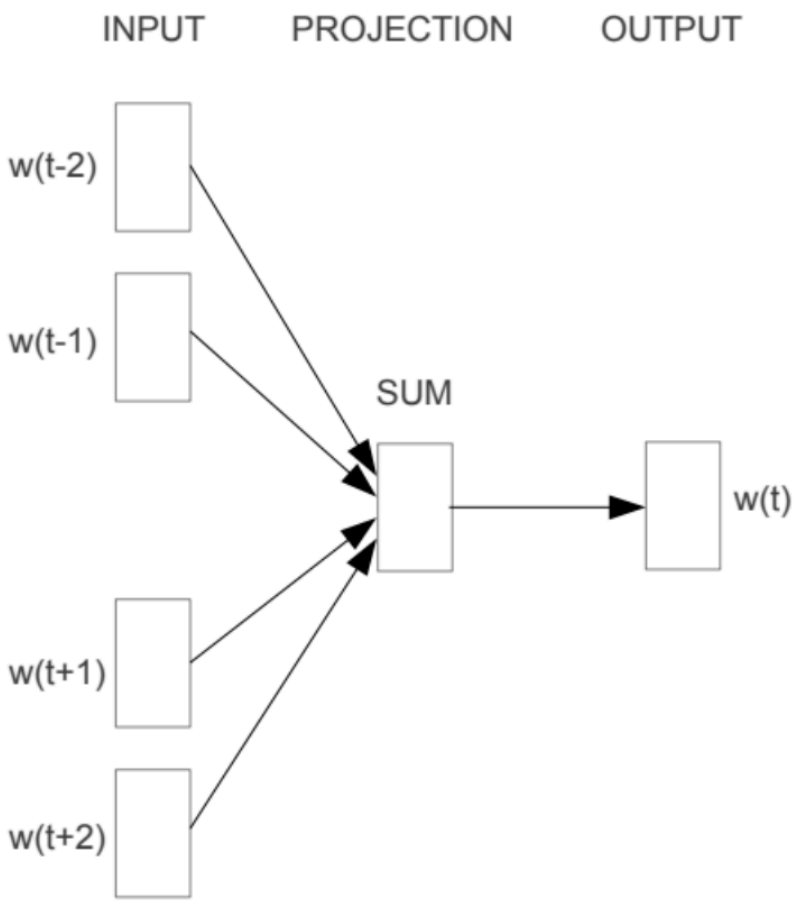
Classification Tasks

- Prediction task is to predict what word will show up at a given ***target*** position (t) in a document given some ***context*** words.
  - Two different ways to conceptualize the problem.

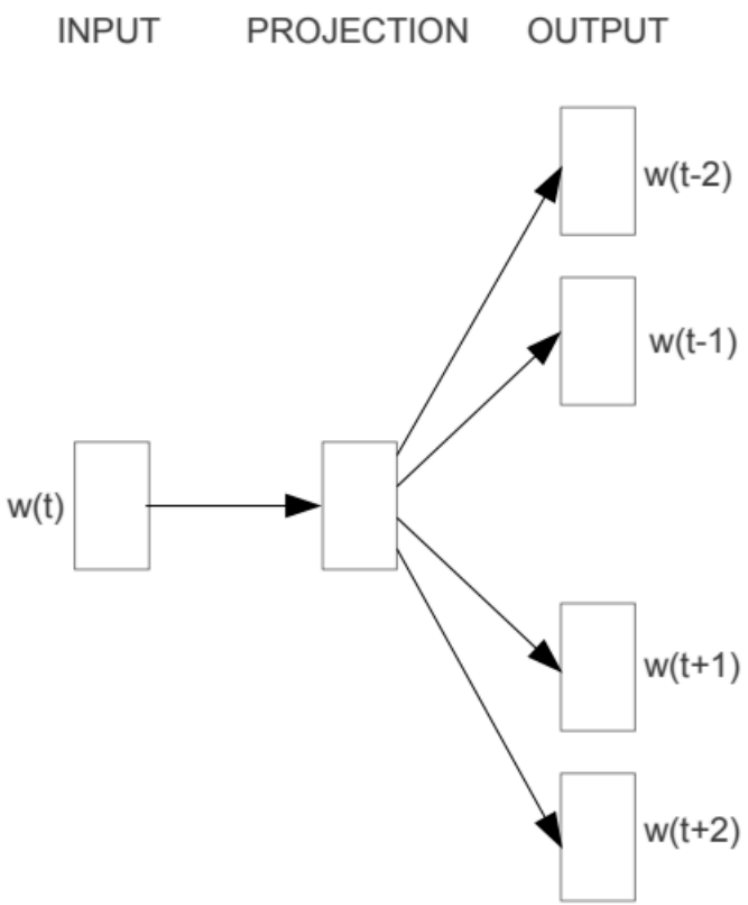
- Continuous Bag of Words (CBOW)
  - Use words that occur near the target word to predict the target word.

- Skip-Grams (SG)
  - Use the target word to predict the context words around that target.
  - Tends to provide better performance for rare words, but is slower.

Classification Tasks

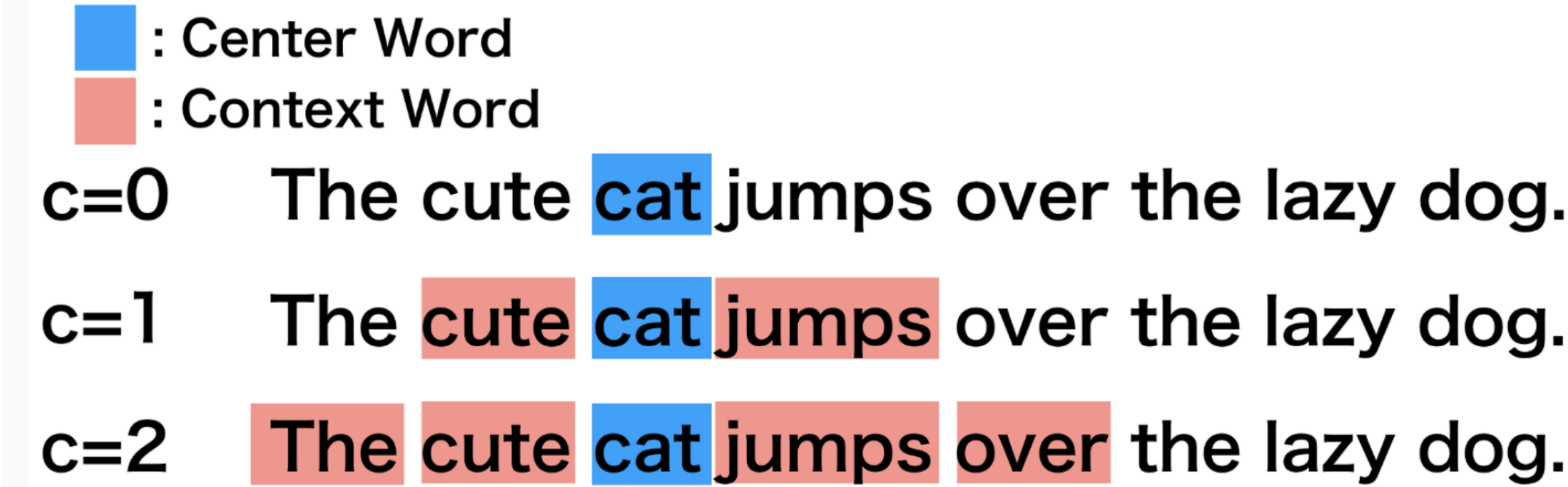


CBOW



Skip-gram

Context Window



Skip-gram training data

- Typically break up the input training data into sentences.
  - Words that are close to other words but separated by sentence boundary can have little to no similarity.
  - Paragraphs with different topics, etc.

Training sentence:

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

c1 c2 target c3 c4

Goal with Skip-gram Approach

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

- (*apricot*, *jam*)
- (*apricot*, *aardvark*)

Return probability that c is a real context word:

$P(+|t,c)$

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

The Dot Product

- Consider two vectors: $\mathbf{a} = [a_1, a_2, \ldots, a_n]$ and $\mathbf{b} = [b_1, b_2, \ldots, b_n]$

- Then to dot product of those two vectors is:

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

- Can serve as a measure of similarity between two vectors as it is inherently related to cosine similarity.

$$\cos(\theta) = \frac{u \cdot v}{||u||\,||v||}$$

How to Generate These Probabilities?

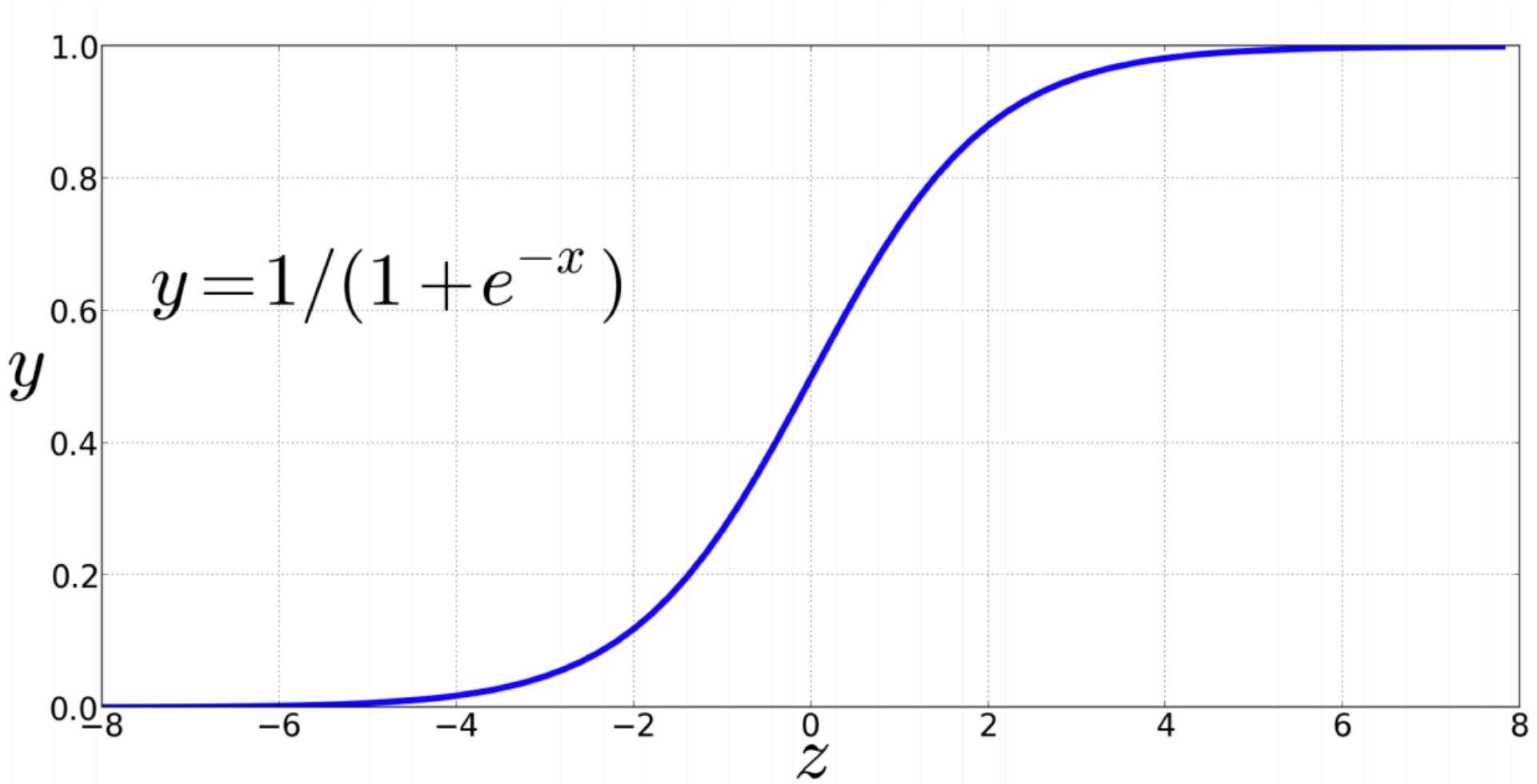
Intuition:

- Words are likely to appear near similar words
- Model similarity with dot-product!
- Similarity(t,c) $\propto t \cdot c$

Problem:

- *Dot product is not a probability!*
  - *(Neither is cosine)*

Logistic Sigmoid Function Gives us Probabilities



Probabilities

- Can generate probabilities of a context word being a real context word for a given target by transforming dot product of context word co-occurring with target word in given context window across all times target word appears.

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

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

- Can roll up this probability over all words in context window.

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

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

Generating Positive and Negative Examples

Training sentence:

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

c1 c2 t c3 c4

positive examples +	
t	c
apricot	tablespoon
apricot	of
apricot	preserves
apricot	or

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

Generating Positive and Negative Examples

Training sentence:

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

c1 c2 t c3 c4

positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	preserves
apricot	or

negative examples - k=2

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

Picking Negative Examples

Could pick w according to their unigram frequency P(w)

More common to chosen then according to $p_\alpha(w)$

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

α= ¾ works well because it gives rare noise words slightly higher probability

To show this, imagine two events p(a)=.99 and p(b) = .01:

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

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

Model Training Objective

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.

Model Training Process

Iterative process.

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:

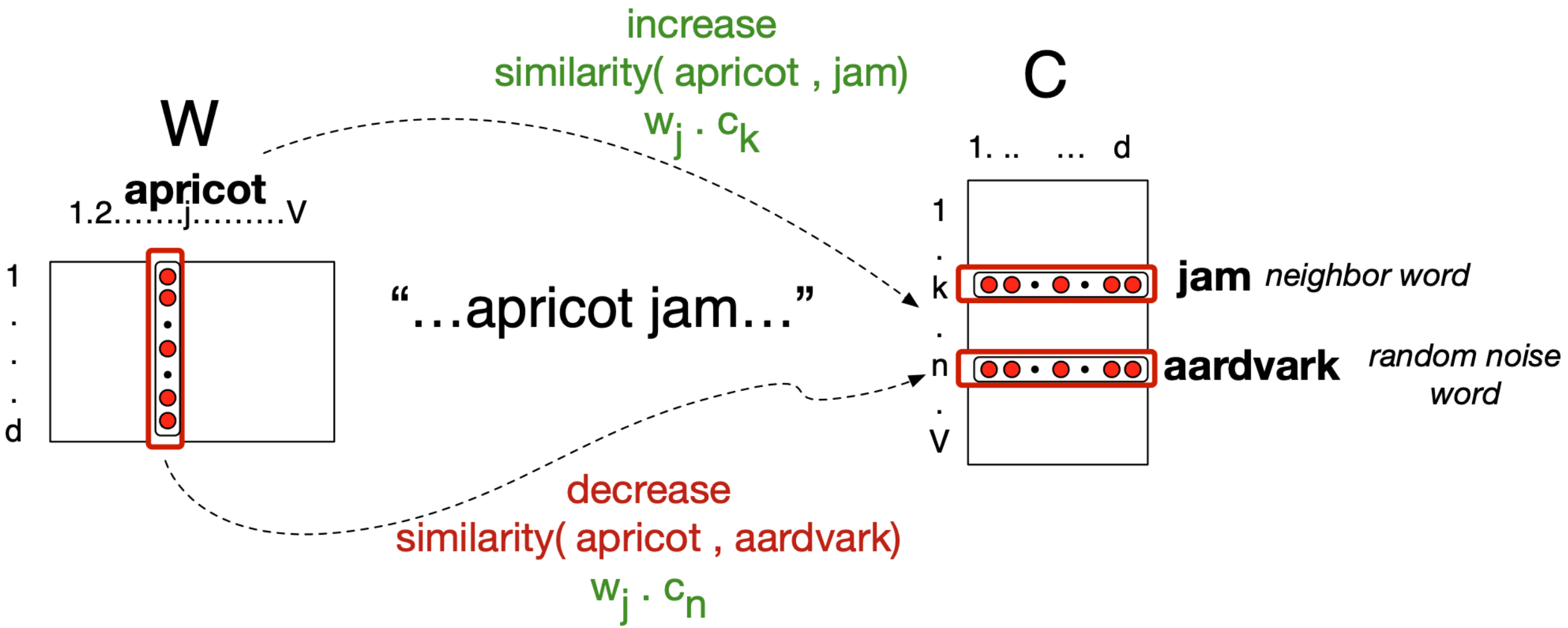
Objective Function

We want to maximize…

$$\sum_{(t,c)\in+} logP(+|t,c) + \sum_{(t,c)\in-} logP(-|t,c)$$

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

Example of Iterative Adjustment



Overview of Word2Vec Training

Start with V random 300-dimensional vectors as initial embeddings

Use logistic regression, the second most basic classifier used in machine learning after naïve bayes

- Take a corpus and take pairs of words that co-occur as positive examples
- Take pairs of words that don't co-occur as negative examples
- Train the classifier to distinguish these by slowly adjusting all the embeddings to improve the classifier performance
- Throw away the classifier code and keep the embeddings.

Context Window Effects Results

- Changing size of context window for training will lead to model identifying different terms as being most similar.

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

- *Sunnydale*
- *Evernight*

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

- *Dumbledore*
- *Malfoy*
- *halfblood*

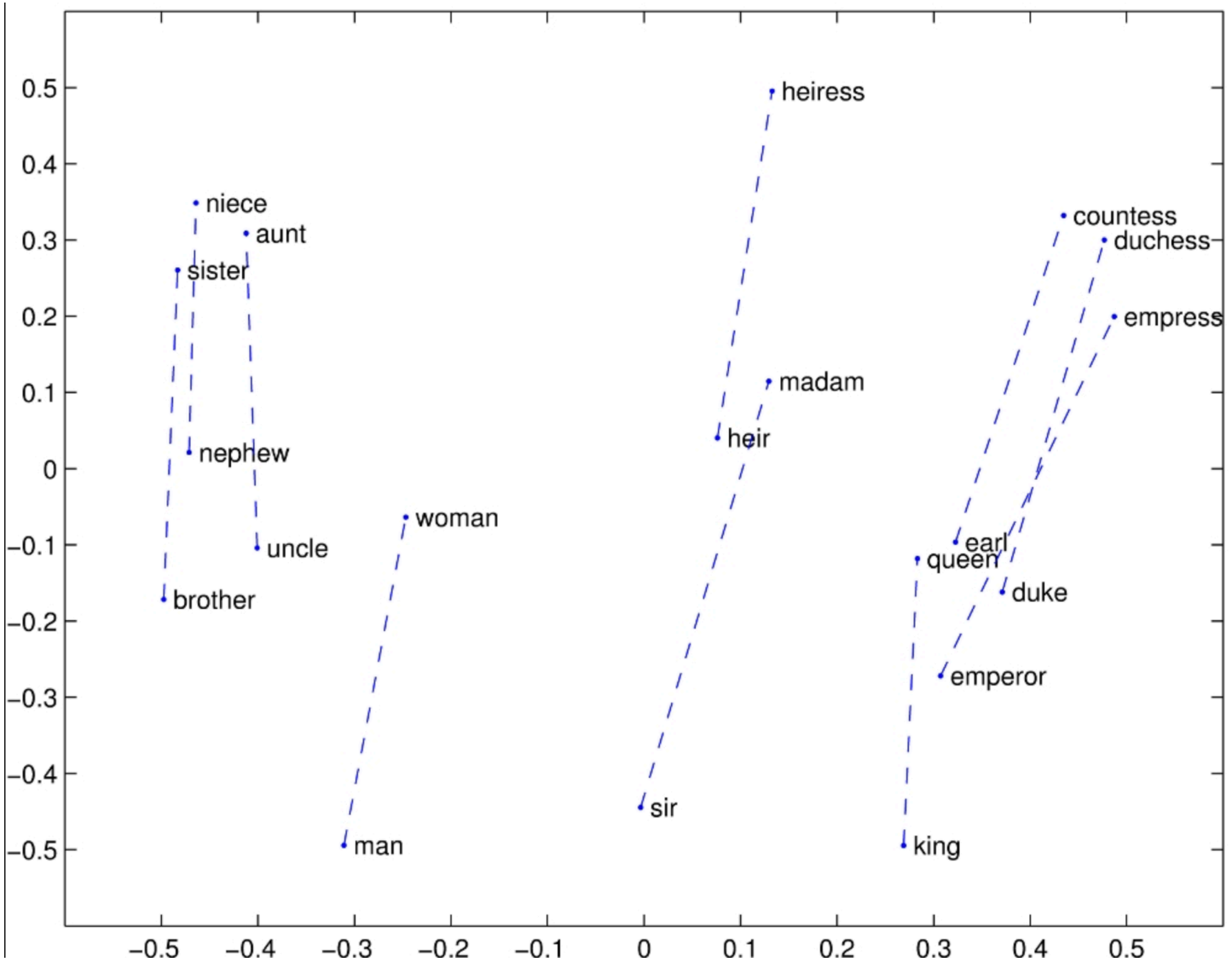
Using Word Embeddings

- Now that we have numbers represented as embedding vectors, we can apply lots of nice mathematical concepts:
  - Similarity between words as distance in the embedding space.
  - Relationships between words as a ray (a direction in the embedding space + a distance)
  - Geometry of concept relations: e.g. analogies.
  - Calibrated embeddings as a way to translate text, concepts.
  - Find different words that share the same meaning when used in different contexts (e.g. legislators and CEOs use different terms to talk about the same regulations).
  - Look at changes in term associations over time, by covariates, to uncover changing word use, linguistic bias, etc.
    - Requires you to train your own embedding model.

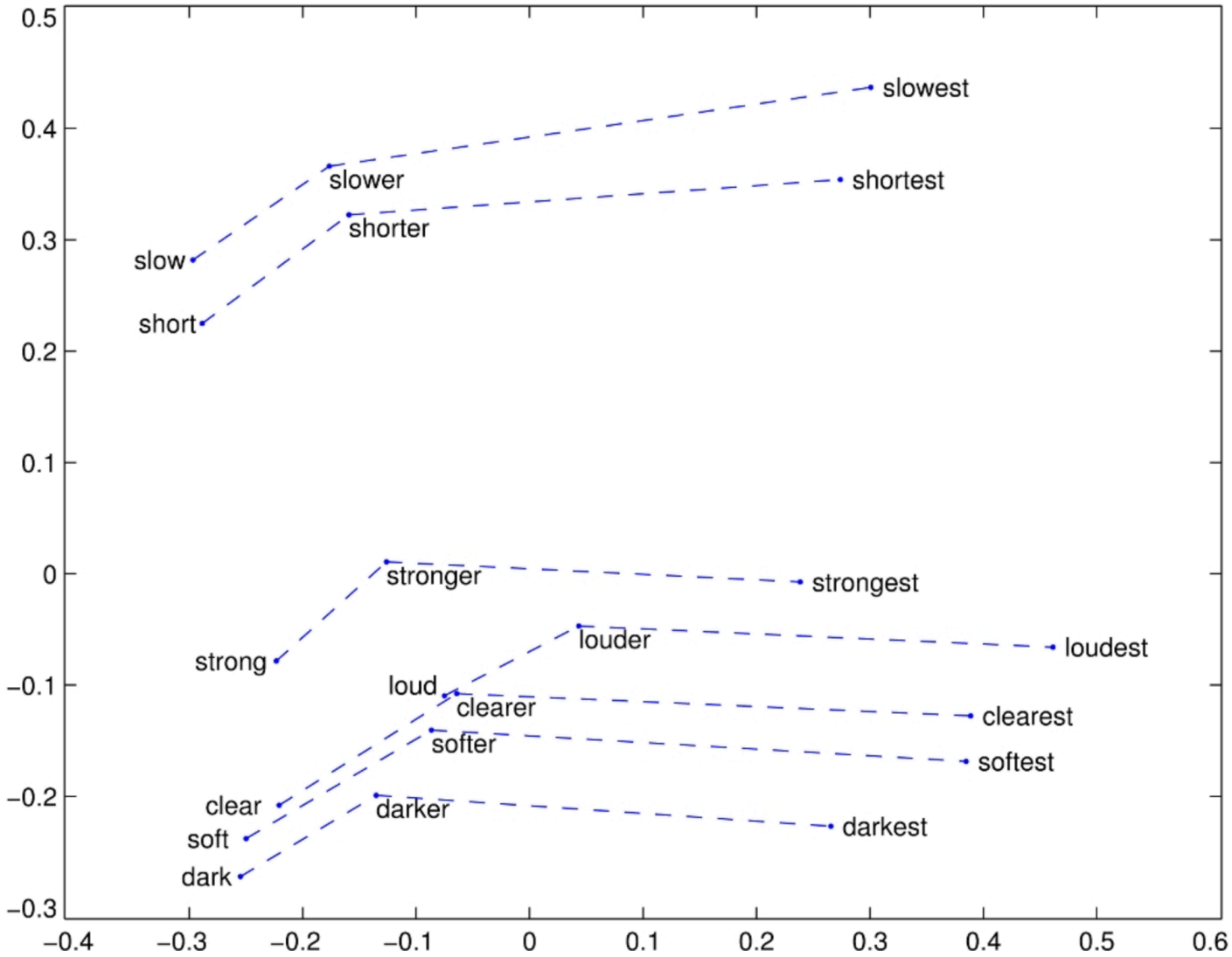
Cosine Similarity

- Input: Let A and B be term vectors as with Euclidean Distance.
- Formula: $$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$
- For text, takes on a value between zero (totally unrelated) to one (meaning exactly terms appear in same proportion).
- Interpreted as angle between term vectors.
- Normalized for document length.

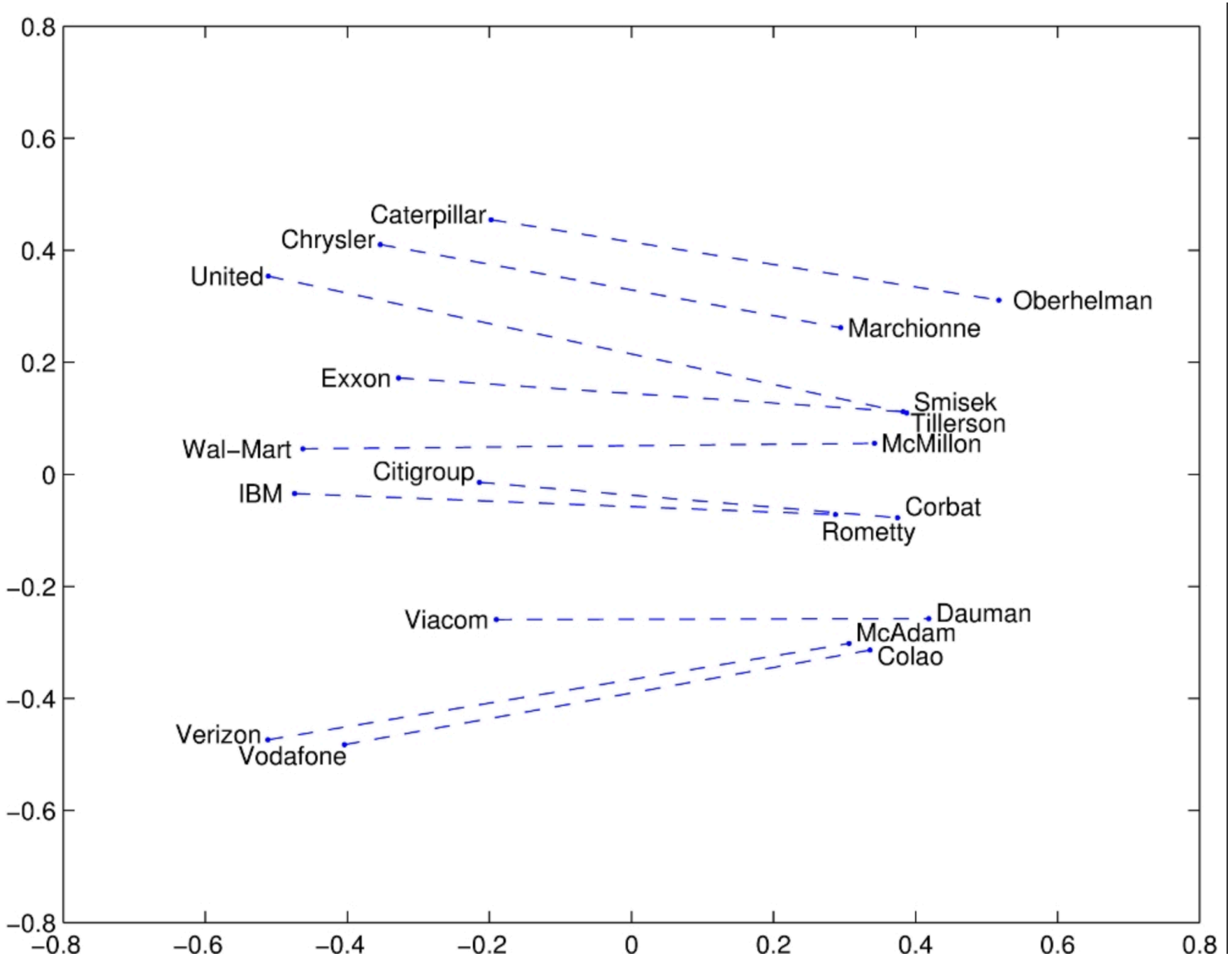
Gender Relationships



Superlative Relationships



Company-CEO Relationships



Solving Analogies

- **US** is to **Canada** as **hamburger** is to **??**
- Relationships are encoded by word vector differences:

$$f(\text{"canada"}) - f(\text{"us"}) = f(\text{"??"}) - f(\text{"hamburger"})$$

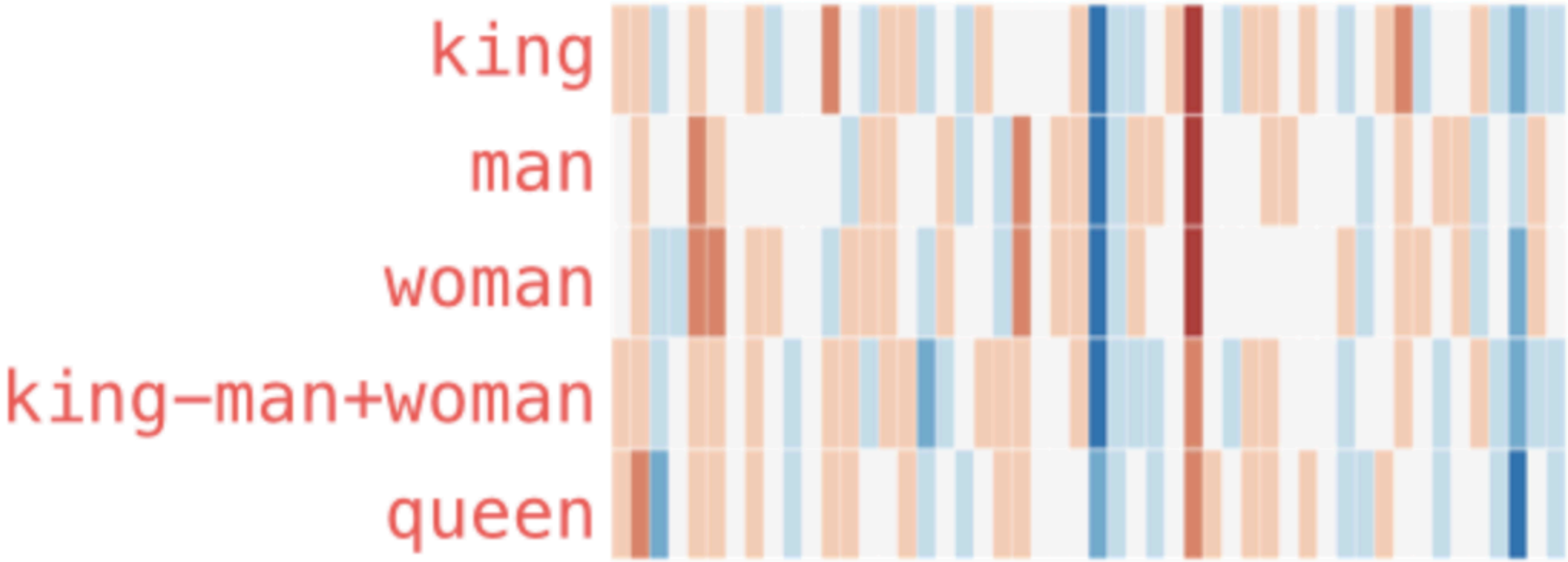
- We can add the relationship encoding vector to a the vector of **hamburger**:

$$f(\text{"hamburger"}) + (f(\text{"canada"}) - f(\text{"us"})) = f(\text{"??"})$$

```
[('poutine', 0.485785305500305),
 ('cheeseburger', 0.4791874885559082),
 ('hotdog', 0.47264325618743896),
 ('australie', 0.4357858002185821 5),
 ('croissant', 0.435305118560791),
 ('hamburgers', 0.4350370168685913),
 ('burger', 0.42210400104522705),
 ('australien', 0.420423686504364),
 ('abendblatt', 0.4082910120487213),
 ('schweiz', 0.404655098915100 1)]
```

Solving Analogies

king − man + woman ~= queen



Social Science Applications

- This is the tricky part…

- Train embeddings on different corpora to estimate differences in word association.
  - Gender and racial bias.
  - Over time.
  - Changes in language use.

- Document similarity, Author similarity/scaling.

- Preprocessing augmentation to combine two different sources of text.

My Take

- Vector space representations of words will become increasingly important over the next few decades with the continued rise of deep learning methods.

- Right now, only a limited set of social science applications can really benefit from using them.

- In many cases, using pre-trained embeddings is the way to go. Need huge amounts of data and processing power to get excellent general purpose results.

- You need a lot more training to know what you are doing when fitting word embedding models. Good opportunity for collaboration.