CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Computer Science

http://www.cit.ie

Natural Language Processing

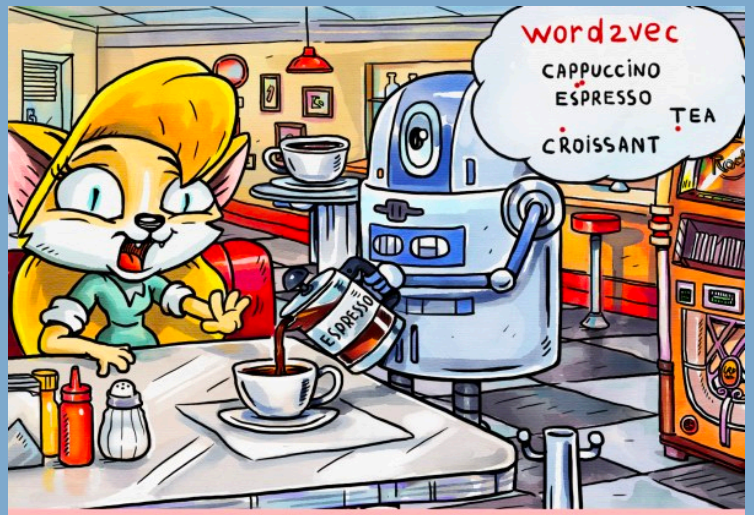
Week5: Dense embeddings

Dr. Haithem Afli

Haithem.afli@cit.ie

@AfliHaithem

2020/2021



- Espresso? But I ordered a cappuccino!
- Don't worry, the cosine distance between them is so small that they are almost the same thing.

Source: datamonsters

Recap

- What is vectorization ?

- How many n_features in the bags of words representation?

- Raw frequency, is it a bad representation?

Recap

- What is vectorization ?

Words need to be encoded as integers or floating point values for use as input to a machine learning algorithms, this process is called **feature extraction** or **vectorization**.

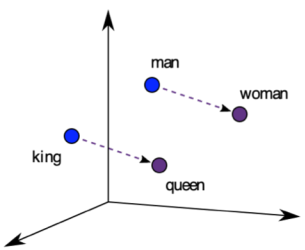
- How many n_features in the bags of words representation?

The bags of words representation implies that **n_features** is the number of **distinct words** in the corpus.

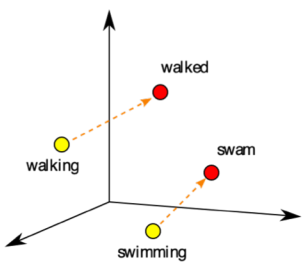
Raw frequency, is it a bad representation?

Content

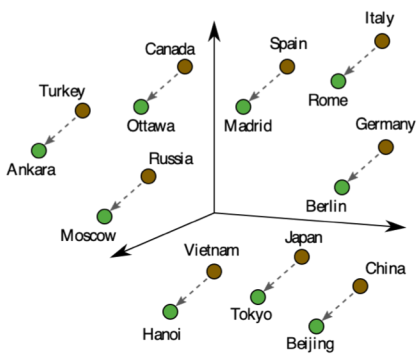
1. **Raw Frequency as Word Representation**
2. Dense Vectors
3. Word2Vec
4. Visualize Word Embedding
5. Word Embedding in Practice



Male-Female

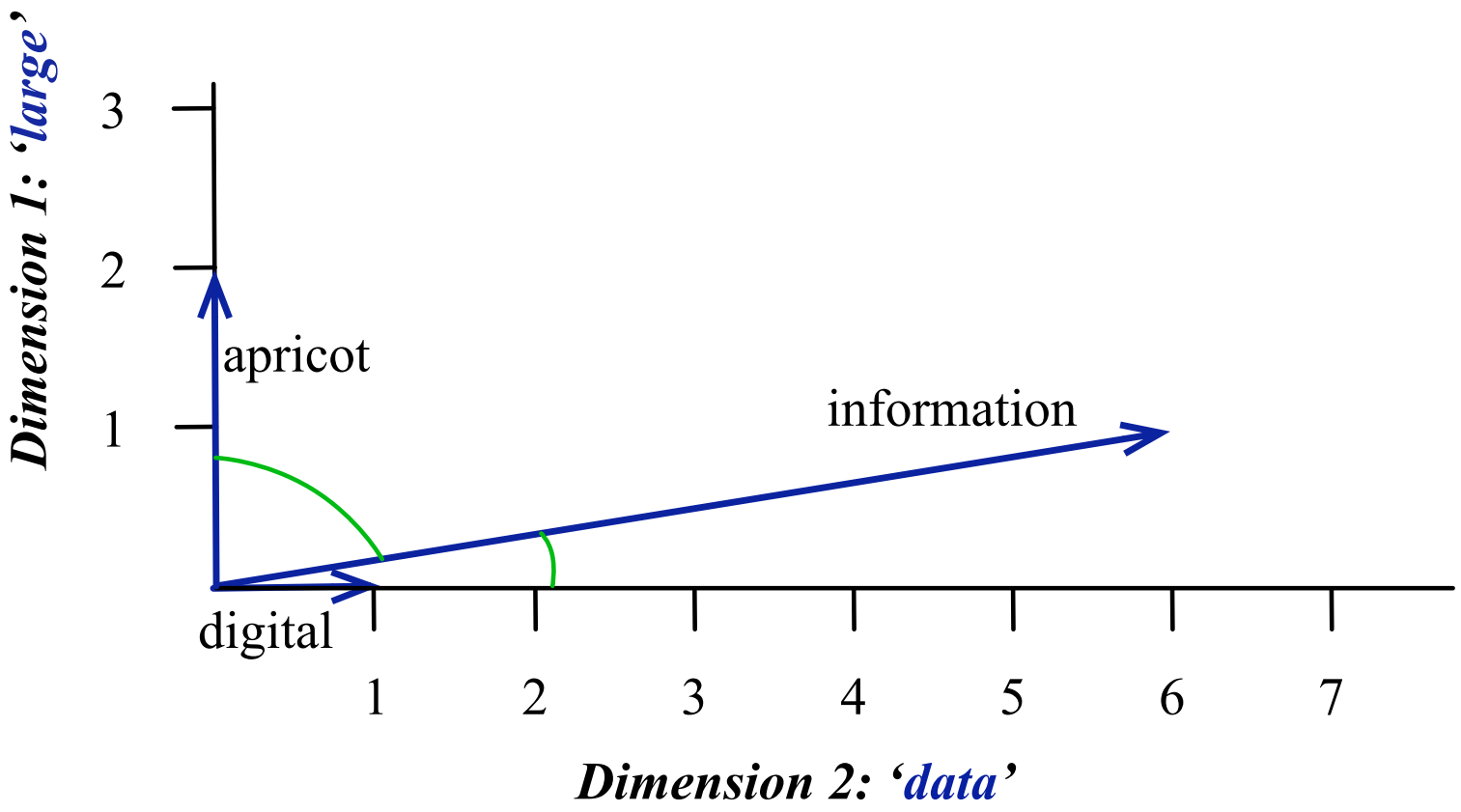


Verb Tense



Country-Capital

Cosine as a similarity metric



Raw frequency is a bad representation!

- Frequency is clearly useful; if *sugar* appears a lot near *apricot*, that's useful information.

But overly frequent words like *the*, *it,* or *they* are not very informative about the context

Need a function that resolves this frequency paradox!

tf-idf: combine two factors

- **tf: term frequency**. frequency count (usually log-transformed):

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

- **Idf: inverse document frequency: tf-**

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

Total # of docs in collection

# of docs that have word i

Words like "the" or "good" have very low idf

tf-idf value for word t in document d:

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

Tf-idf is a sparse representation

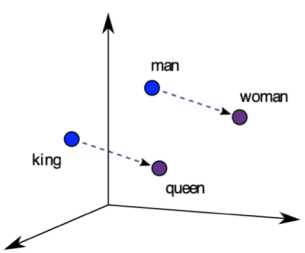
- tf-idf vectors are
  - **long** (length |V|= 20,000 to 50,000)
  - **sparse** (most elements are zero)

We need

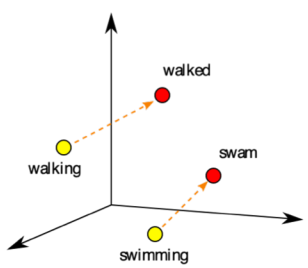
- vectors which are
  - **short** (length 50-1000)
  - **dense** (most elements are non-zero)

Content

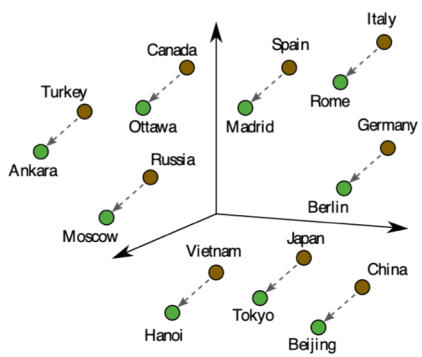
1. Raw Frequency as Word Representation
2. **Dense Vectors**
3. Word2Vec
4. Visualize Word Embedding
5. Word Embedding in Practice



Male-Female



Verb Tense



Country-Capital

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 are distinct dimensions
      - a word with *car* as a neighbor and a word with *automobile* as a neighbor should be similar, but aren't
  - **In practice, they work better**

Dense embeddings you can download!

- **Word2vec** (Mikolov et al.)

https://code.google.com/archive/p/word2vec

- **Fasttext**

http://www.fasttext.cc/

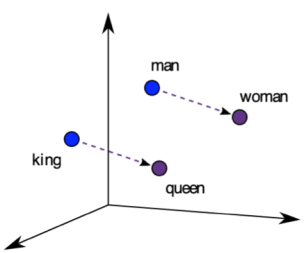
- **Glove** (Pennington, Socher, Manning)

http://nlp.stanford.edu/projects/glove/

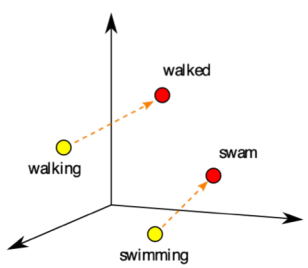
Discussion

Content

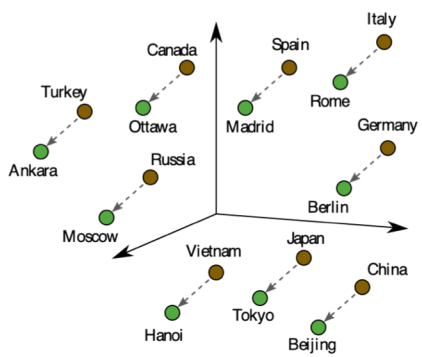
1. Raw Frequency as Word Representation
2. Dense Victors
3. **Word2Vec**
4. Visualize Word Embedding
5. Word Embedding in Practice



Male-Female



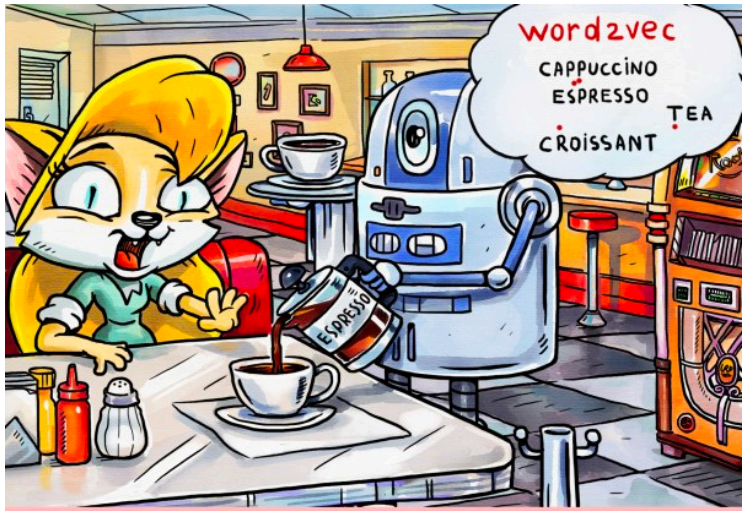
Verb Tense



Country-Capital

Word2vec

- Popular embedding method
- Very fast to train
- Code available on the web
- Idea: **predict** rather than **count**



- Espresso? But I ordered a cappuccino!
- Don't worry, the cosine distance between them is so small that they are almost the same thing. **Source: datamonsters**

Word2vec

- 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 as implicitly supervised training data!

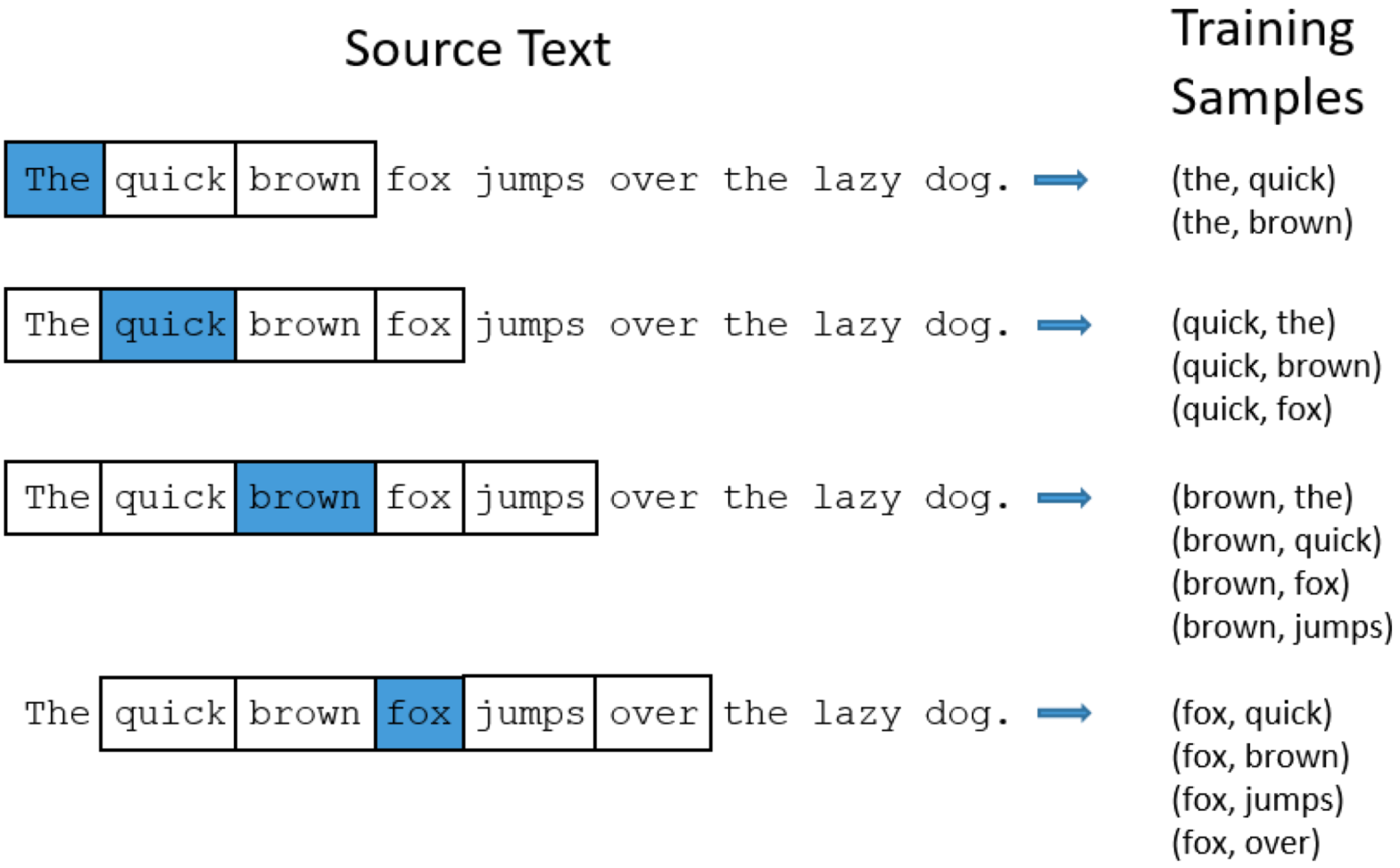
- A word *s* near *apricot*
  - Acts as gold ‘correct answer’ to the question
  - “Is word *w* likely to show up near *apricot*?”
- No need for hand-labeled supervision
- The idea comes from **neural language modeling**
  - Bengio et al. (2003)
  - Collobert et al. (2011)

Word2Vec: **Skip-Gram** Task

- Word2vec provides a variety of options.

  Let's do "skip-gram with negative sampling" (SGNS)

Skip-gram algorithm



Discussion

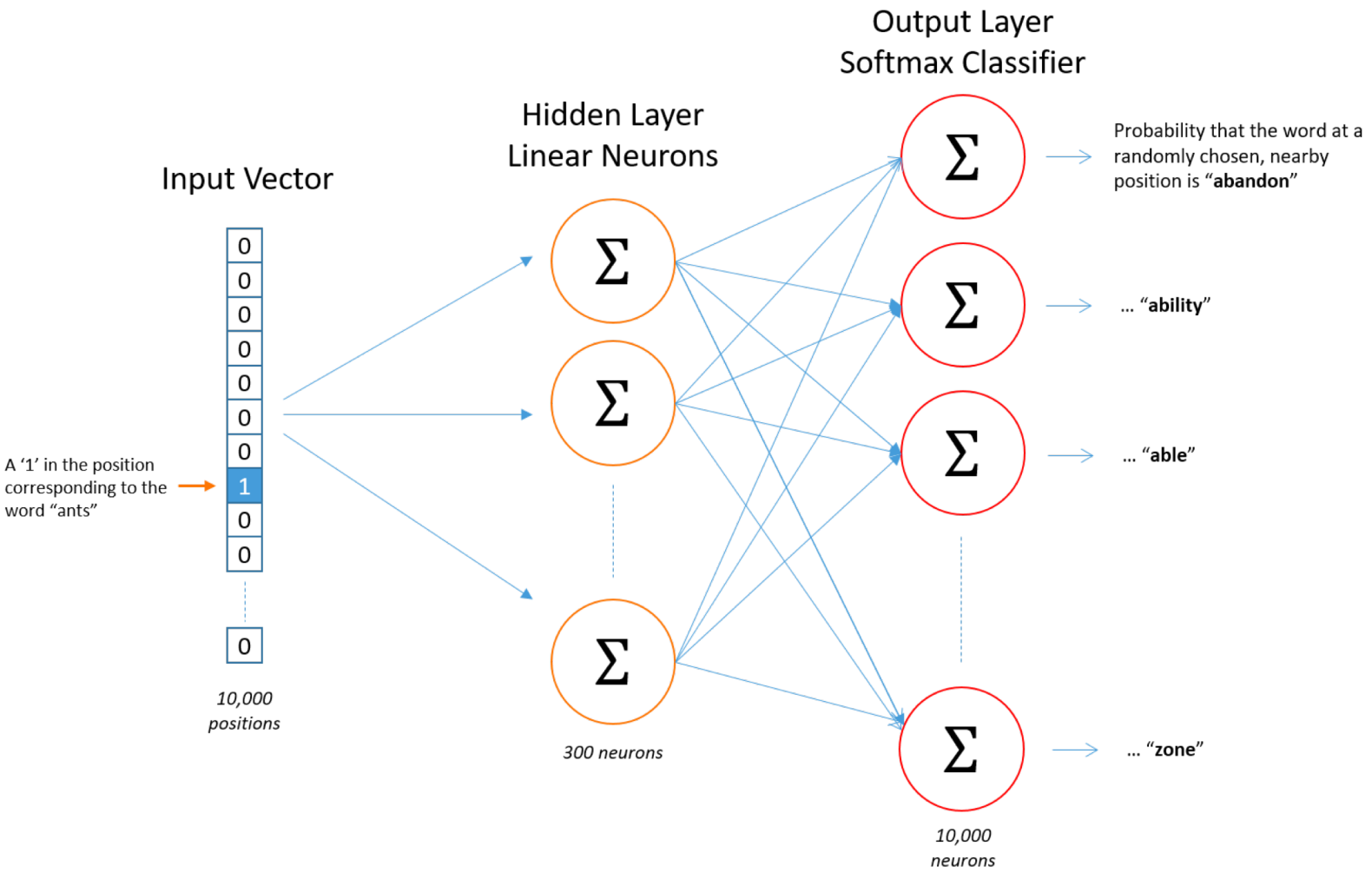
Skip-gram algorithm

- The example shows some of the training samples (word pairs) we would take from the sentence “The quick brown fox jumps over the lazy dog.”
- A small window size of 2 used just for the example.
- The word highlighted in blue is the input word.

Skip-gram algorithm

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

Skip-Gram Training Data



Skip-Gram Training Data

- Training sentence:
- ... lemon, a tablespoon of **apricot** jam a pinch ...
- c1 c2 target c3 c4

Asssume context words are those in +/- 2 word window

Skip-Gram Goal

- Given a tuple (t,c) = target, context
  - (*apricot*, *jam*)
  - (*apricot*, *aardvark*)
- Return probability that c is a real context word:

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

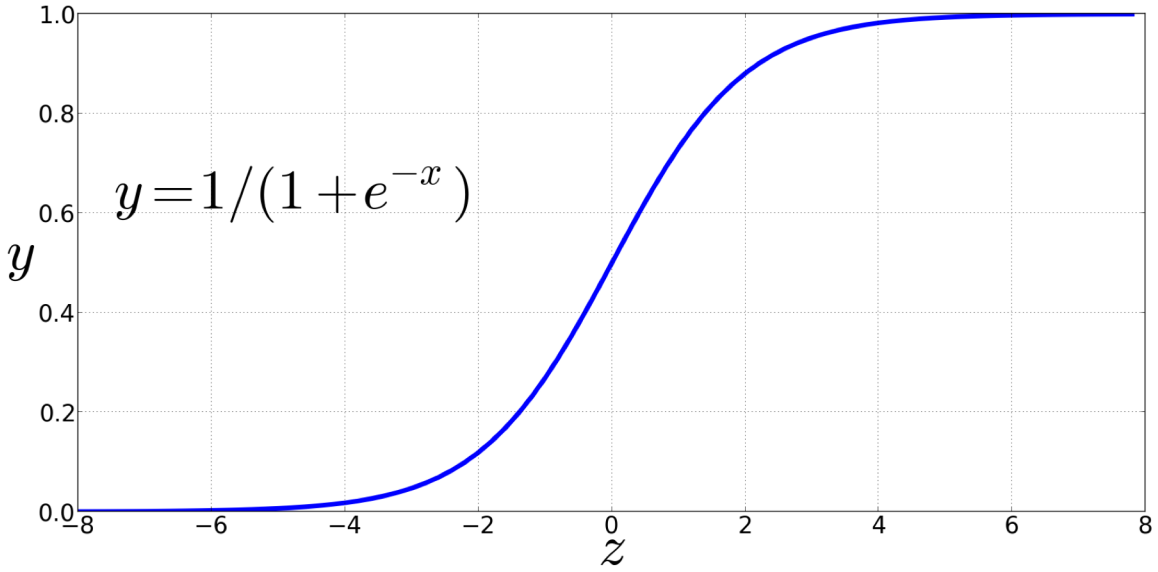
How to compute p(+|t,c)?

- 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)*

Turning dot product into a probability

- The sigmoid lies between 0 and 1:

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



Turning dot product into a probability

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

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

$$= \frac{e^{-t\cdot c}}{1+e^{-t\cdot c}}$$

For all the context words:

- Assume all context words are independent

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

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

Skip-Gram Training Data

- Training sentence:
- ... lemon, a tablespoon of **apricot** jam a pinch ...
- c1 c2 t c3 c4

- Training data: input/output pairs centering on *apricot*
- Asssume a +/- 2 word window

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	preserves
apricot	or

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

Skip-Gram Training

- Training sentence:
- ... lemon, a tablespoon of **apricot** jam a pinch ...
- c1 c2 t c3 c4

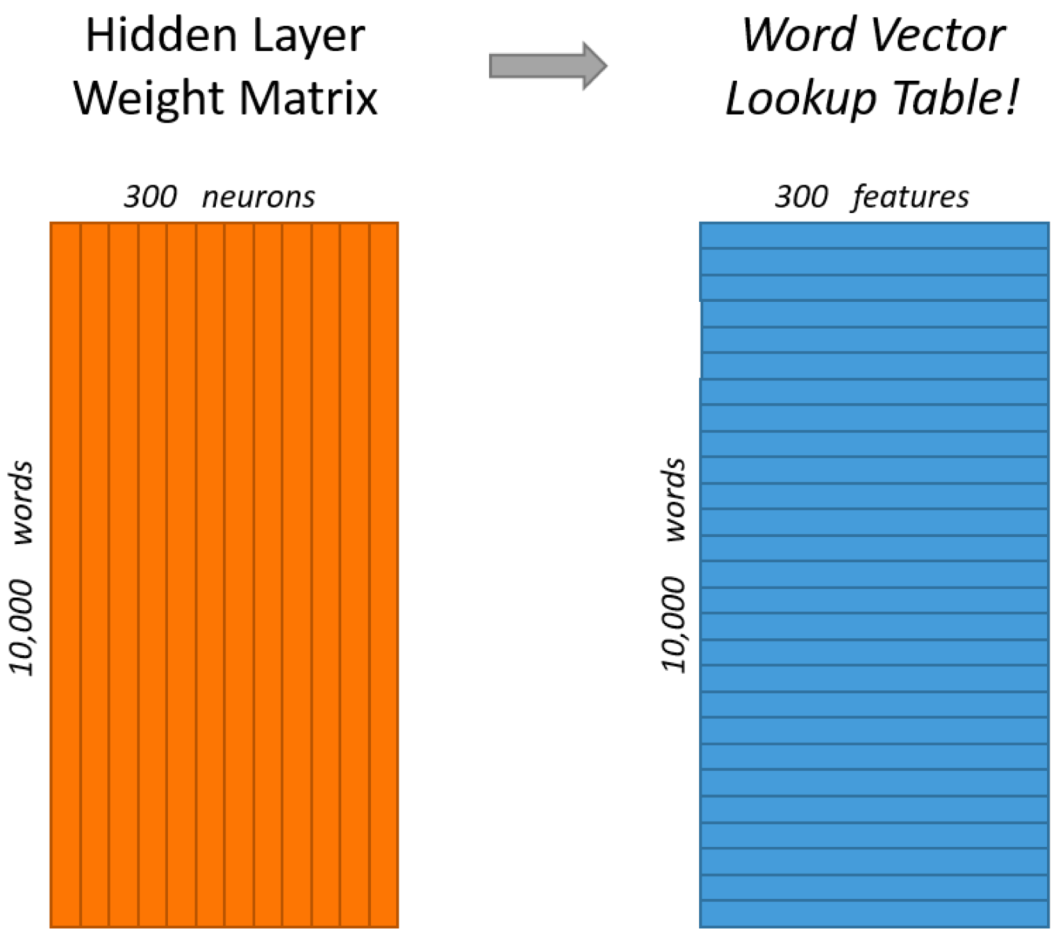
k=2

positive examples +		**negative examples -**			
t	c	t	c	t	c
apricot	tablespoon	apricot	aardvark	apricot	twelve
apricot	of	apricot	puddle	apricot	hello
apricot	preserves	apricot	where	apricot	dear
apricot	or	apricot	coaxial	apricot	forever

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.

300 features is what Google used in their published model trained on the Google news dataset


Hidden Layer
Weight Matrix
Word Vector
Lookup Table!
300 neurons
10,000 words
300 features
10,000 words

Objective Criteria

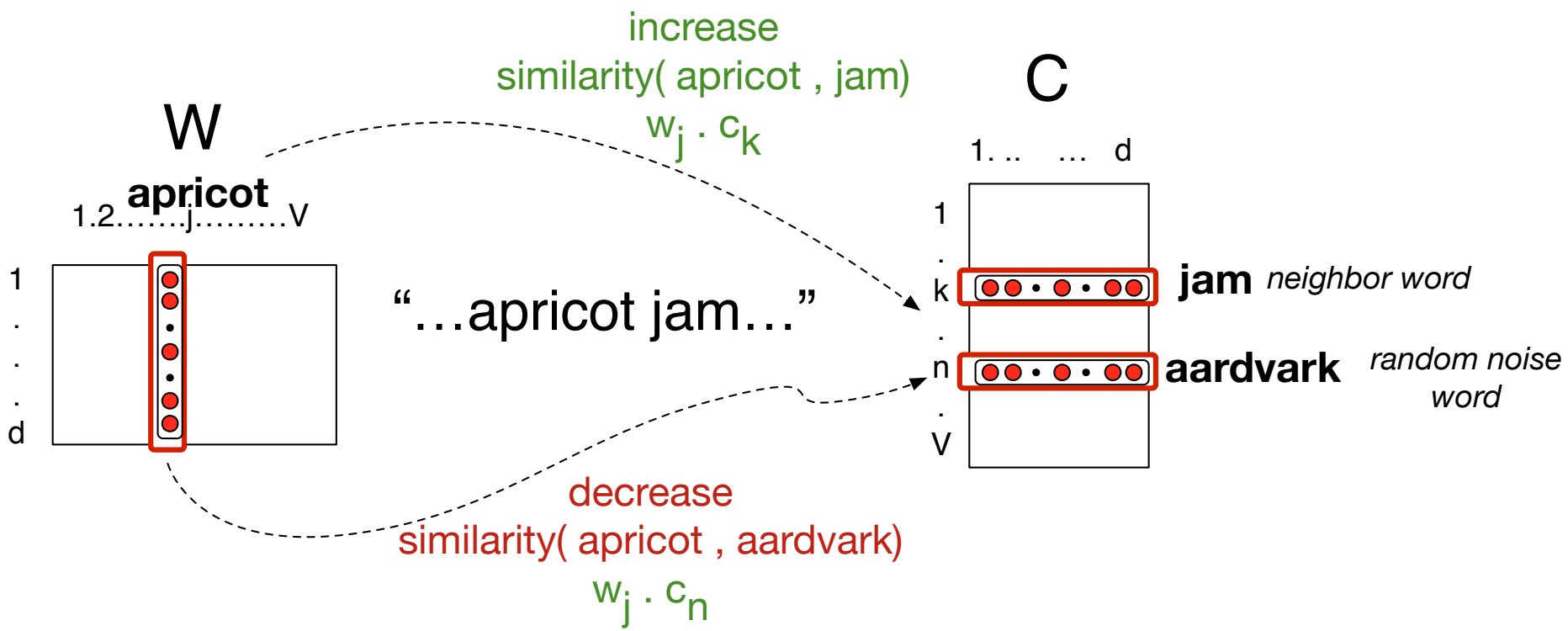
- We want to maximize…

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

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

Focusing on one target word t:

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


increase
similarity(apricot , jam)
$w_j \cdot c_k$
W
apricot
1.2.....j.....V
1
.
.
.
d
"…apricot jam…"
C
1. ..
... d
1
.
k
.
n
.
V
jam
neighbor word
aardvark
random noise word
decrease
similarity(apricot , aardvark)
$w_j \cdot c_n$

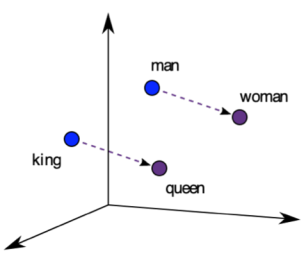
Properties of embeddings

Similarity depends on window size C

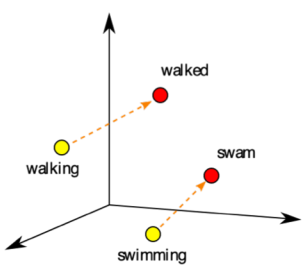
- C = ±2 The nearest words to *Hogwarts:*
  - *Sunnydale*
  - *Evernight*
- C = ±5 The nearest words to *Hogwarts:*
  - *Dumbledore*
  - *Malfoy*
  - *halfblood*

Content

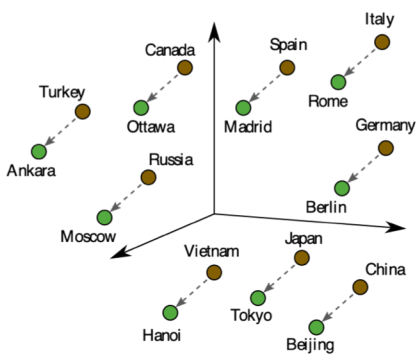
1. Raw Frequency as Word Representation
2. Dense Vectors
3. Word2Vec
4. **Visualize Word Embedding**
5. Word Embedding in Practice



Male-Female



Verb Tense



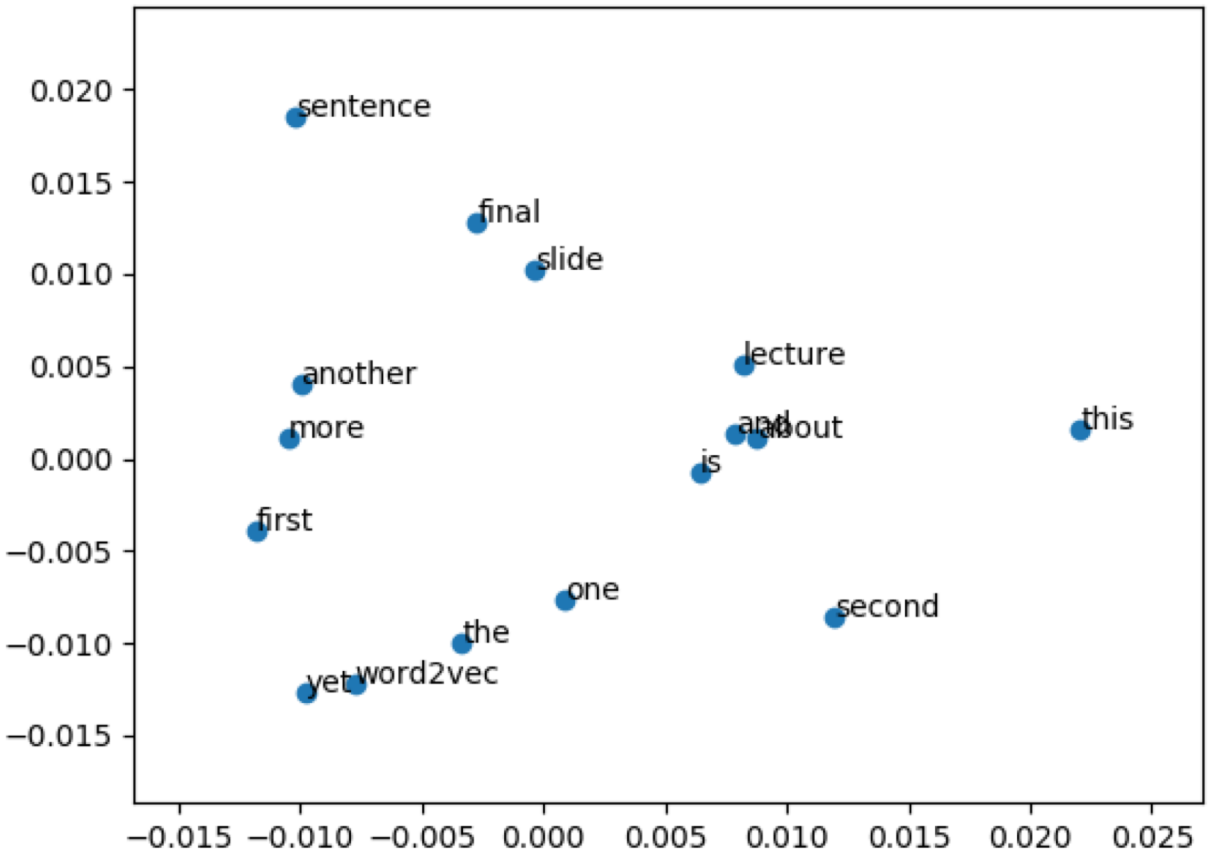
Country-Capital

Discussion

Develop Word2Vec Embedding


```
from gensim.models import Word2Vec
# define training data
# define training data
sentences = [['this', 'is', 'the', 'second', 'lecture', 'about', 'word2vec'],
             ['this', 'is', 'the', 'first', 'slide'],
             ['yet', 'another', 'slide'],
             ['one', 'more', 'sentence'],
             ['and', 'the', 'final', 'sentence']]
# train model
model = Word2Vec(sentences, min_count=1)
# summarize the loaded model
print(model)
# summarize vocabulary
words = list(model.wv.vocab)
print(words)
# access vector for one word
print(model['sentence'])
# save model
model.save('model.bin')
# load model
new_model = Word2Vec.load('model.bin')
print(new_model)
```

Visualize Word Embedding



- Training your own word vectors may be the best approach for a given NLP problem.
- But it can take a long time, a fast computer with a lot of RAM and disk space, and perhaps some expertise in finessing the input data and training algorithm.
- An alternative is to simply use an existing pre-trained word embedding.

- GoogleNews-vectors-negative300.bin.gz. https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit?usp=sharing

Load Google's Word2Vec Embedding

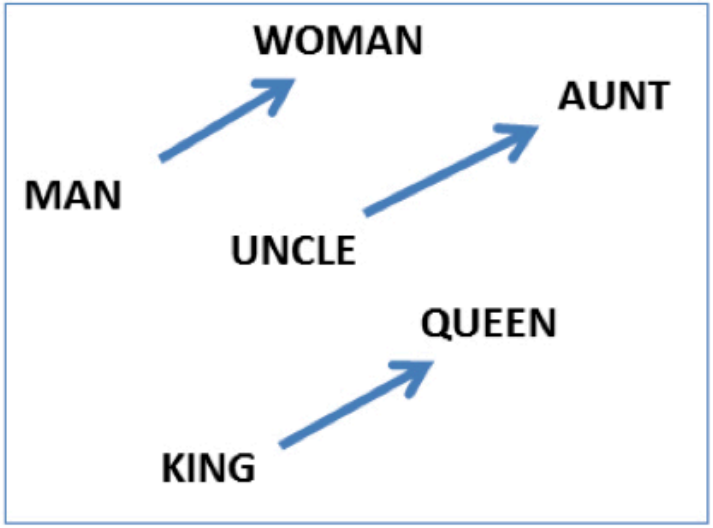
```
1  from gensim.models import KeyedVectors
2  # load the google word2vec model
3  filename = 'GoogleNews-vectors-negative300.bin'
4  model = KeyedVectors.load_word2vec_format(filename, binary=True)
5  # calculate: (king - man) + woman = ?
6  result = model.most_similar(positive=['woman', 'king'], negative=['man'], topn=1)
7  print(result)
```

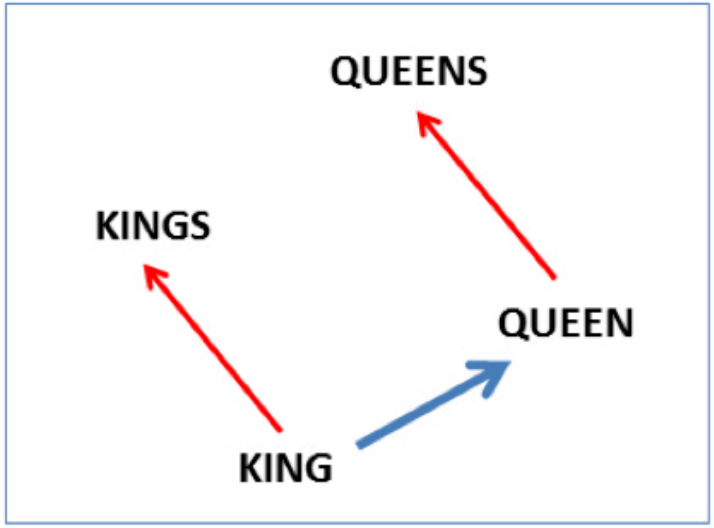
- You can do is do a little linear algebra arithmetic with words queen = (king - man) + woman
- So we run the code of : (king - man) + woman = ?
- Result: [('queen', 0.7118192315101624)]

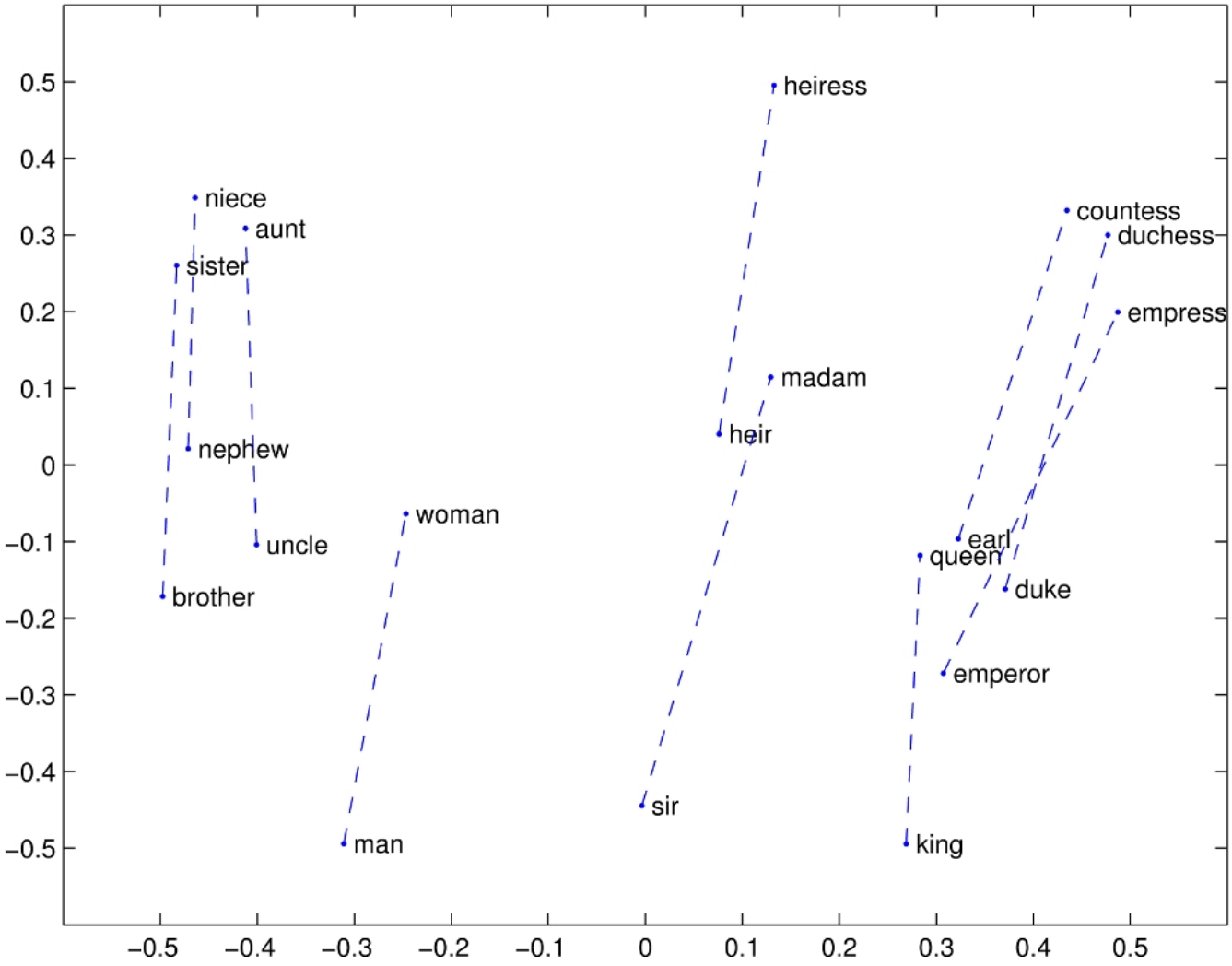
Analogy: Embeddings capture relational meaning!

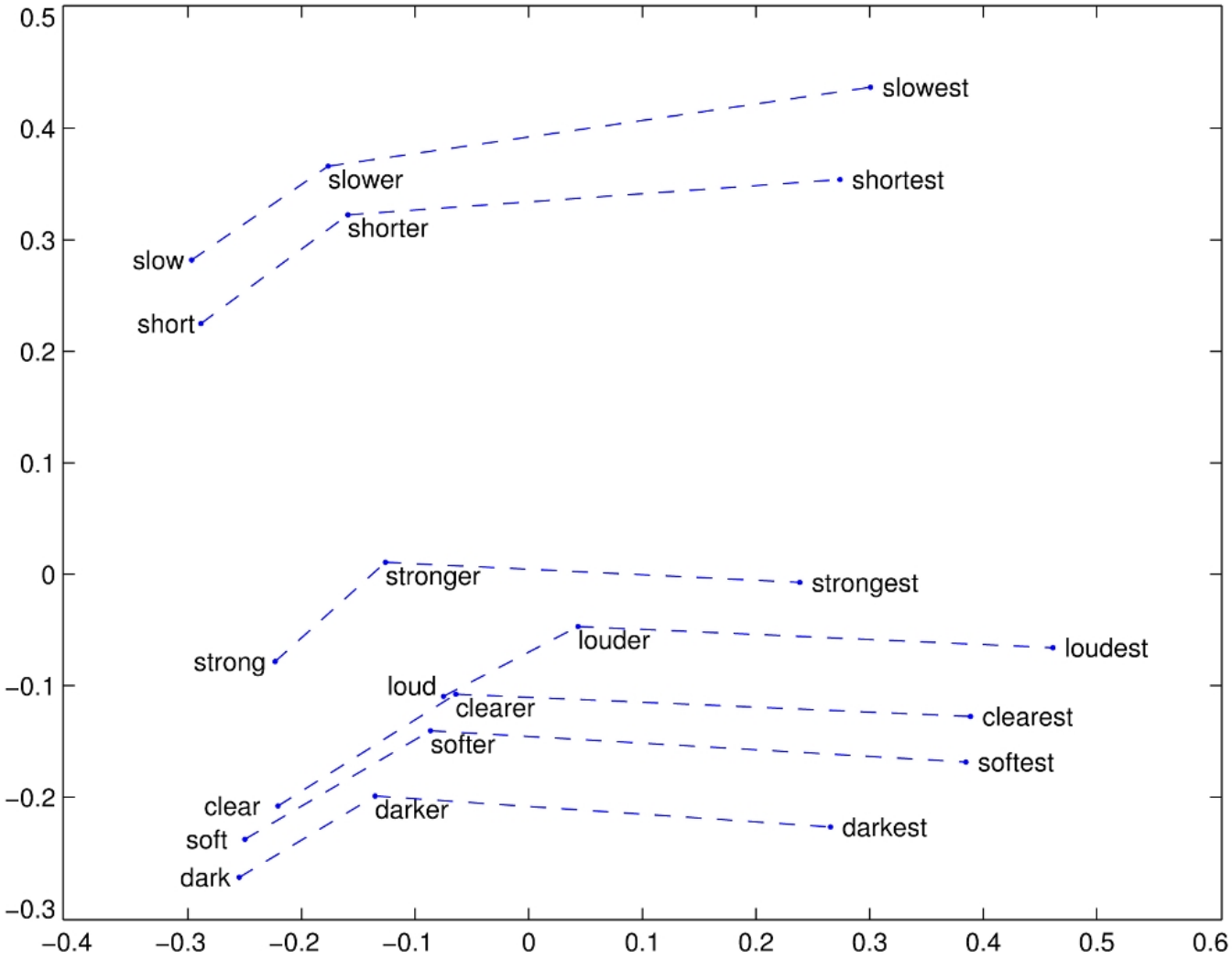
vector(*'king'*) - vector(*'man'*) + vector(*'woman'*) ≈ vector('queen')

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



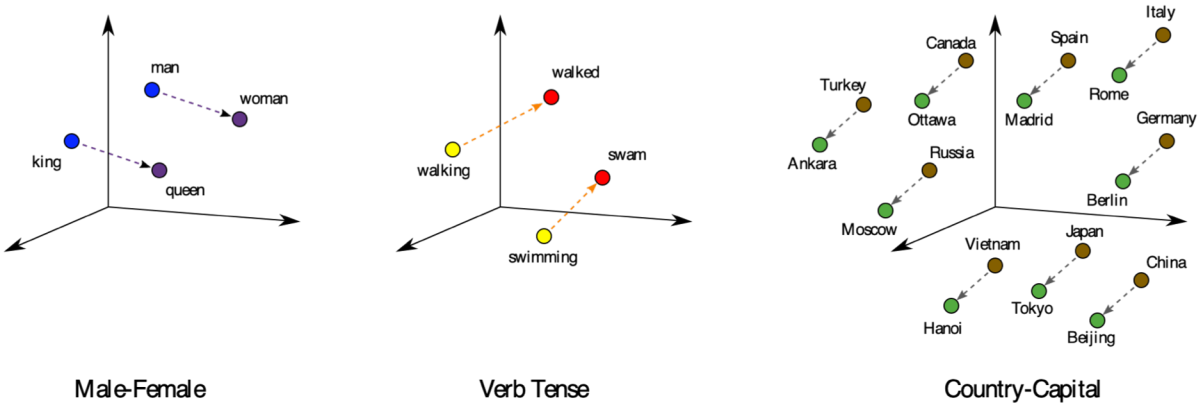



heiress
niece
aunt
countess
duchess
sister
empress
madam
nephew
heir
woman
earl
queen
uncle
duke
brother
emperor
sir
man
king
0.5
0.4
0.3
0.2
0.1
0
−0.1
−0.2
−0.3
−0.4
−0.5
−0.5
−0.4
−0.3
−0.2
−0.1
0
0.1
0.2
0.3
0.4
0.5


0.5
0.4
0.3
0.2
0.1
0
−0.1
−0.2
−0.3
−0.4
−0.3
−0.2
−0.1
0
0.1
0.2
0.3
0.4
0.5
0.6
slowest
slower
shortest
shorter
slow
short
stronger
strongest
louder
loudest
strong
loud
clearer
clearest
softer
softest
clear
darker
darkest
soft
dark

Content

1. Raw Frequency as Word Representation
2. Dense Vectors
3. Word2Vec
4. Visualize Word Embedding
5. **Word Embedding in Practice**

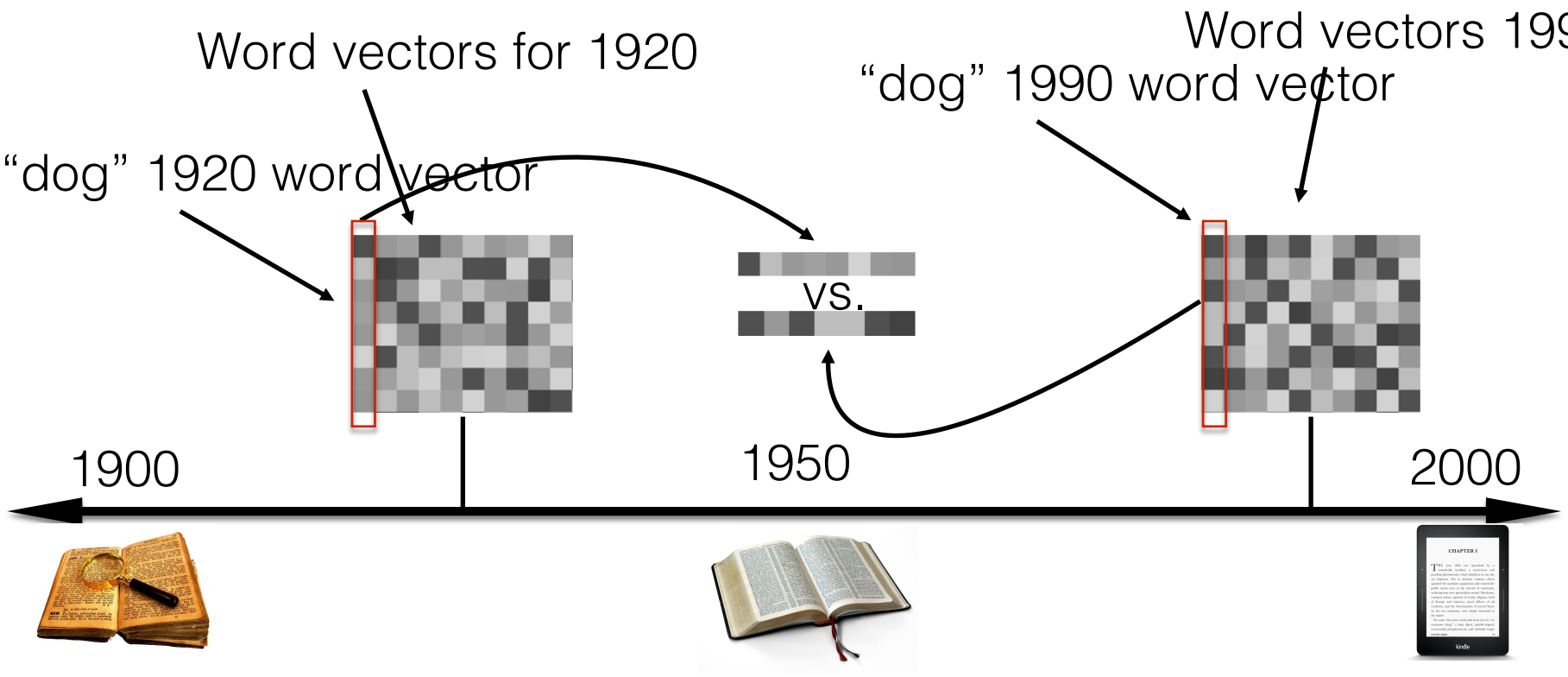


Male-Female Verb Tense Country-Capital

Embeddings can help study word history!

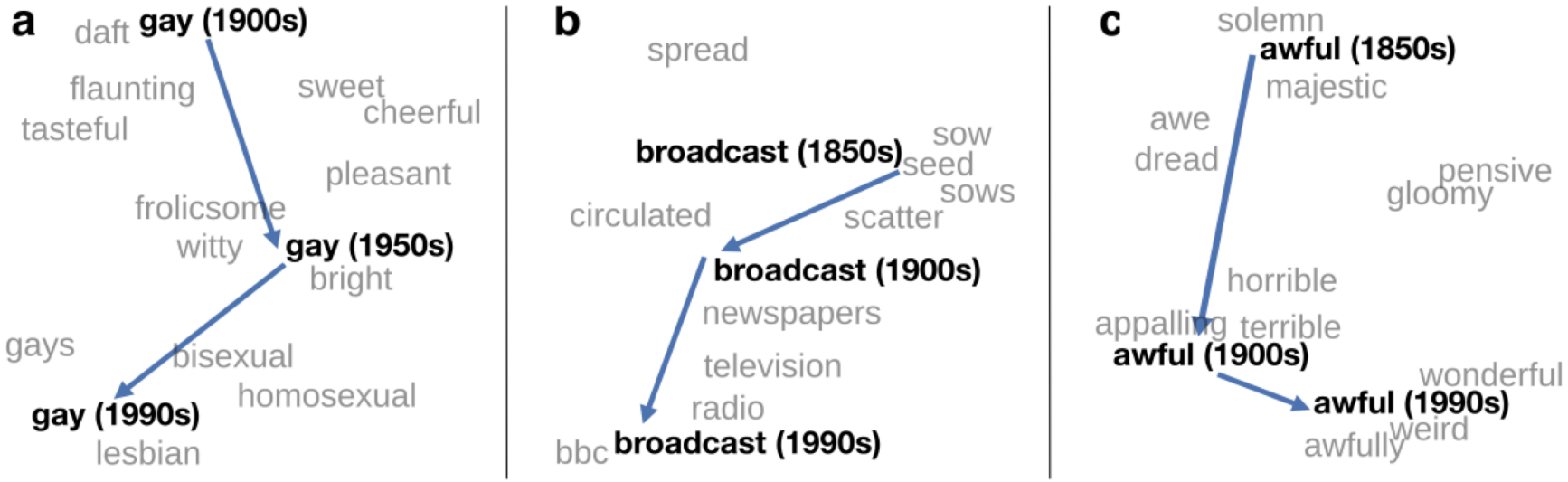
- Train embeddings on old books to study changes in word meaning!!

Diachronic word embeddings for studying language change!



Visualizing changes

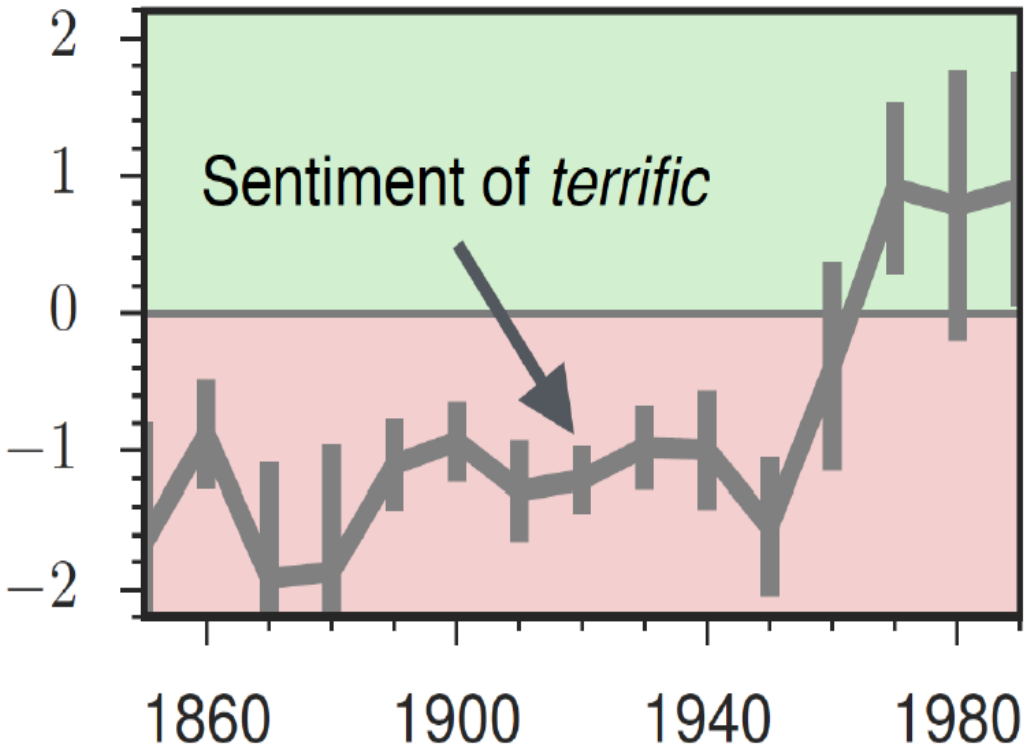
Project 300 dimensions down into 2



~30 million books, 1850-1990, Google Books data

The evolution of sentiment words

Negative words change faster than positive words



Embeddings reflect cultural bias

Bolukbasi, Tolga, Kai-Wei Chang, James Y. Zou, Venkatesh Saligrama, and Adam T. Kalai. "Man is to computer programmer as woman is to homemaker? debiasing word embeddings." In *Advances in Neural Information Processing Systems*, pp. 4349-4357. 2016.

- Ask “Paris : France :: Tokyo : x”
  - x = Japan
- Ask “father : doctor :: mother : x”
  - x = nurse
- Ask “man : computer programmer :: woman : x”
  - x = homemaker

Embeddings reflect cultural bias

Caliskan, Aylin, Joanna J. Bruson and Arvind Narayanan. 2017. Semantics derived automatically from language corpora contain human-like biases. Science 356:6334, 183-186.

- Implicit Association test (Greenwald et al 1998): How associated are
  - concepts (*flowers*, *insects*) & attributes (*pleasantness*, *unpleasantness*)?
  - Studied by measuring timing latencies for categorization.
- Psychological findings on US participants:
  - African-American names are associated with unpleasant words (more than European-American names)
  - Male names associated more with math, female names with arts
  - Old people's names with unpleasant words, young people with pleasant words.
- Caliskan et al. replication with embeddings:
  - African-American names (*Leroy, Shaniqua*) had a higher GloVe cosine with unpleasant words (*abuse, stink, ugly*)
  - European American names (*Brad, Greg, Courtney*) had a higher cosine with pleasant words (*love, peace, miracle*)
- Embeddings reflect and replicate all sorts of pernicious biases.

Embeddings as a window onto history

Garg, Nikhil, Schiebinger, Londa, Jurafsky, Dan, and Zou, James (2018). Word embeddings quantify 100 years of gender and ethnic stereotypes. *Proceedings of the National Academy of Sciences*, *115*(16), E3635–E3644

- Use the Hamilton historical embeddings
- The cosine similarity of embeddings for decade X for occupations (like teacher) to male vs female names
  - Is correlated with the actual percentage of women teachers in decade X

History of biased framings of women

Garg, Nikhil, Schiebinger, Londa, Jurafsky, Dan, and Zou, James (2018). Word embeddings quantify 100 years of gender and ethnic stereotypes. *Proceedings of the National Academy of Sciences*, *115*(16), E3635–E3644

- Embeddings for competence adjectives are biased toward men
  - *Smart, wise, brilliant, intelligent, resourceful, thoughtful, logical, etc.*
- This bias is slowly decreasing

Embeddings reflect ethnic stereotypes over time

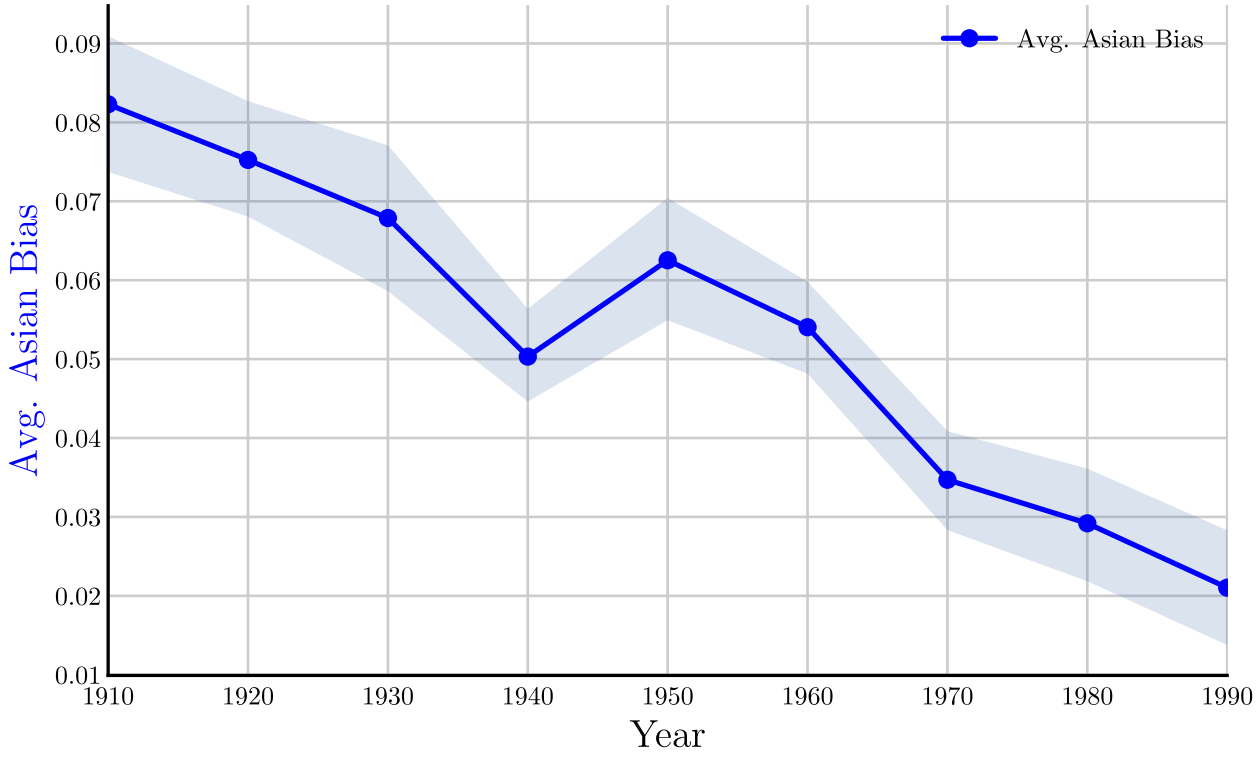
Garg, Nikhil, Schiebinger, Londa, Jurafsky, Dan, and Zou, James (2018). Word embeddings quantify 100 years of gender and ethnic stereotypes. *Proceedings of the National Academy of Sciences*, *115*(16), E3635–E3644

- Princeton trilogy experiments
- Attitudes toward ethnic groups (1933, 1951, 1969) scores for adjectives
  - *industrious, superstitious, nationalistic*, etc
- Cosine of Chinese name embeddings with those adjective embeddings correlates with human ratings.

Change in linguistic framing 1910-1990

Garg, Nikhil, Schiebinger, Londa, Jurafsky, Dan, and Zou, James (2018). Word embeddings quantify 100 years of gender and ethnic stereotypes. *Proceedings of the National Academy of Sciences*, *115*(16), E3635–E3644

Change in association of Chinese names with adjectives framed as "othering" (*barbaric*, *monstrous*, *bizarre*)



Changes in framing: adjectives associated with Chinese

Garg, Nikhil, Schiebinger, Londa, Jurafsky, Dan, and Zou, James (2018). Word embeddings quantify 100 years of gender and ethnic stereotypes. *Proceedings of the National Academy of Sciences*, *115*(16), E3635–E3644

1910	1950	1990
Irresponsible	Disorganized	Inhibited
Envious	Outrageous	Passive
Barbaric	Pompous	Dissolute
Aggressive	Unstable	Haughty
Transparent	Effeminate	Complacent
Monstrous	Unprincipled	Forceful
Hateful	Venomous	Fixed
Cruel	Disobedient	Active
Greedy	Predatory	Sensitive
Bizarre	Boisterous	Hearty

More applications

- https://dl.acm.org/doi/pdf/10.1145/3293339.3293346

Word Embedding based Semantic Cross-Lingual Document Alignment in Comparable Corpora

Debasis Ganguly[1], Haithem Afli[2], Dwaipayan Roy[3]
[1] IBM Research, Dublin, Ireland
debasis.ganguly1@ie.ibm.com
[2] ADAPT Centre, Cork Institute of Technology, Cork , Ireland
haithem.afli@cit.ie
[3] Indian Statistical Institute, Kolkata, India
dwaipayan_r@isical.ac.in

ABSTRACT

Crosslingual information retrieval (CLIR) finds its application in aligning documents across comparable corpora. However, traditional CLIR, due to the term independence assumption, cannot consider the semantic similarity between the constituent words of the candidate pairs of documents in two different languages. Moreover, traditional CLIR models score a document by aggregating only the weights of the constituent terms that *match* with those of the query, while the other non-matching terms of the document do not significantly contribute to the similarity function. Word vector embedding allows the provision to model the semantic distances between terms by the application of standard distance metrics between their corresponding real valued vectors. This paper develops a word vector embedding based CLIR model that uses the average distances between the embedded word vectors of the source and target language documents to rank candidate document pairs. Our experiments with the WMT bilingual document alignment dataset reveal that the word vector based similarity significantly improves the recall of crosslingual document alignment in comparison to the classical language modeling based CLIR.

The main advantage of a CLIR-based method is that it is computationally much faster than an MT-based method, because: a) it is not required to translate each document from the source language to the target language; and b) it is not required to compute the inter-document similarities between all pairs of documents from the source and the target languages, a quadratic time operation. The main challenge with the application of the CLIR method for the purpose of crosslingual document alignment is to address the underlying semantic match between the comparable documents in the source and the target languages. With this motivation, we investigate the use of word-vector embedding for the purpose of developing a semantic matching model for predicting the candidate alignments more effectively.

A characteristic of the word-vector embedding obtained by a method, such as 'word2vec' [14], is that the relative distances between the embedded word vectors is often a good estimate of the semantic distance between two words. The objective function in 'word2vec' learned with a recurrent neural network (RNN)-based approach on a large volume of text ensures that the inner product between the vector representation of two words u and v is high (or in other words, the vectors are similar) if v is likely to occur in the

Conclusion

- **Embeddings** = vector models of meaning
  - More fine-grained than just a string or index
  - Especially good at modeling similarity/analogy
    - Just download them and use cosines!!
  - Can use sparse models (tf-idf) or dense models (word2vec, GLoVE)
  - Useful in practice but know they encode cultural stereotypes

Discussion

Some content was adapted from Speech and Language Processing - Jurafsky and Martin

CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Computer Science

http://www.cit.ie

Thank you

Haithem. afli@cit.ie

@AfliHaithem