Introduction to Word Embeddings

Overview

- What is a word embedding?
- What sorts of word embeddings are there?
  - Count Vectors
  - TF-IDF Vectors
  - Word2Vec
  - GloVe
- How do we evaluate word embeddings?

What is a word embedding?

What is a word embedding?

- Before we can even think about building a model to make some predictions or analysis with text data, we need a way to represent words in a numeric way that computers can understand.

- How do we represent “cat” as a collection of values that we can do some number crunching on?

- A ***word embedding*** is a method or technique of converting words in our vocabulary into vectors that we can later feed in to down-stream machine learning and data science tasks.

Count Vectors

Count Vectors

- Consider a *corpus* of documents { D_1 , D_2 , … , D_N }
- Suppose that after looking over all of our documents, we determine that we have V terms in our total vocabulary.
- With this, we can create a *document-term matrix* with shape N x V which describes how many times a given term appears in a document.

Count Vectors

Consider this tiny corpus:

- I like NLP
- Learning machine learning is great
- NLP is great

The document-term matrix is:

	I	like	NLP	machine	learning	is	great
D_1	1	1	1	0	0	0	0
D_2	0	0	0	1	2	1	1
D_3	0	0	1	0	0	1	1

Count Vectors

- With the document-term matrix, the document/word vectors are just the rows/columns!

	I	like	NLP	machine	learning	is	great
D_1	1	1	1	0	0	0	0
D_2	0	0	0	1	2	1	1
D_3	0	0	1	0	0	1	1

- The document vector for "I like NLP" is $(1, 1, 1, 0, 0, 0)^T$
- The word vector for "NLP" is $(1, 0, 1)^T$

- What are some pros and cons of this approach?

Term Frequency – Inverse Document Frequency

Term Frequency – Inverse Document Frequency

- Term Frequency – Inverse Document Frequency (TF-IDF) is another popular method
- TF-IDF builds upon raw count vectors in two ways
  - Using term frequency instead of raw counts (per document)
  - Also takes into account term frequency across the entire corpus
- *Stop words* like “a”, “the”, “and”, etc. will appear in nearly every document. They add a lot of noise and not much information, so we would like to reduce their influence.

TF-IDF

- TF-IDF has two components
- Term Frequency: How frequently does each term appear in a document

$$TF(t,d) = \frac{\#\ of\ times\ term\ t\ appears\ in\ document\ d}{Total\ number\ of\ terms\ in\ document\ d}$$

- Inverse Document Frequency: Across all documents, how often does a term appear?

$$IDF(t,D) = \ln\left(1 + \frac{Total\ number\ of\ documents, D}{Number\ of\ documents\ containing\ term\ t}\right)$$

- TF-IDF just combines these two!

$$TFIDF(t,d,D) = TF(t,d) \times IDF(t,D)$$

TF-IDF

- Pros
  - Simple to compute
  - Provides a simple basis for comparing the similarity of documents
  - Great tool for a lexical description of text

- Cons
  - TF-IDF is based on a Bag-of-Words approach, and can’t capture positional information.
  - Cannot capture semantic information

Word2Vec

Word2Vec

- The methods we have discussed work really well in some contexts, but there are several important ways they could be improved
  - Count/Frequency methods produce vectors which are very high-dimensional and very sparse
  - Count/Frequency methods really struggle to capture semantic relationships between words
- But how do words actually *get* their meaning?
- According to the *Distributional Hypothesis*, words that are used in similar contexts tend to have similar meaning


“
You shall know a word by the company it keeps.
”
– J. R. Firth

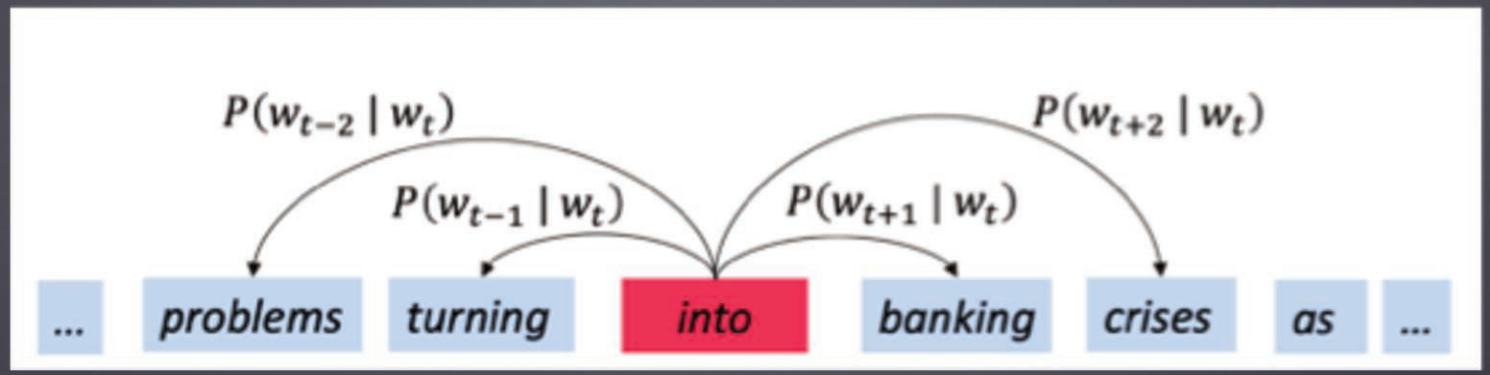
Word2Vec

- Inspired by the Distributional Hypothesis, researchers started turning towards *language models* as a means of producing dense word vectors

- A language model is just a way for us to assign probabilities to a sequence of words
  - “The cat sat on the mat” would have a relatively high probability (It is valid English)
  - “on cat the The sat mat” would have a relatively low probability (It is invalid English)

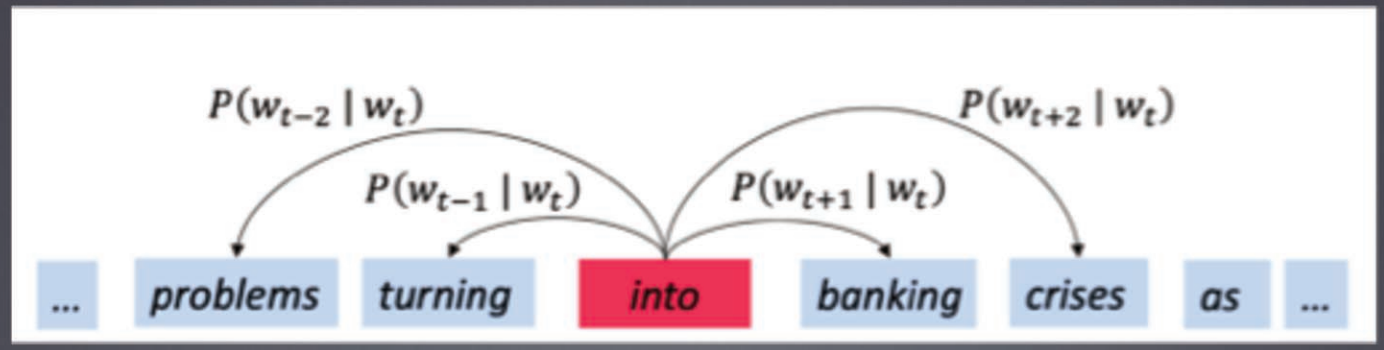
Word2Vec

- Some examples of language models
  - Unigram model: $P(w_1 w_2 \ldots w_n) = \prod_{i=1}^{n} P(w_i)$
  - Bigram model: $P(w_1 w_2 \ldots w_n) = \prod_{i=2}^{n} P(w_i \mid w_{i-1})$
- In contrast to these, word2vec uses the *skip-gram* architecture as a language model.



http://web.stanford.edu/class/cs224n/lectures/lecture2.pdf

Word2Vec



http://web.stanford.edu/class/cs224n/lectures/lecture2.pdf

- For each word in our corpus (t=1 … T) Predict the 2*m* surrounding words!
  - $J'(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \neq 0}} P\left(w_{t+j} \mid w_t ; \theta\right)$ (θ denotes the word2vec model parameters)

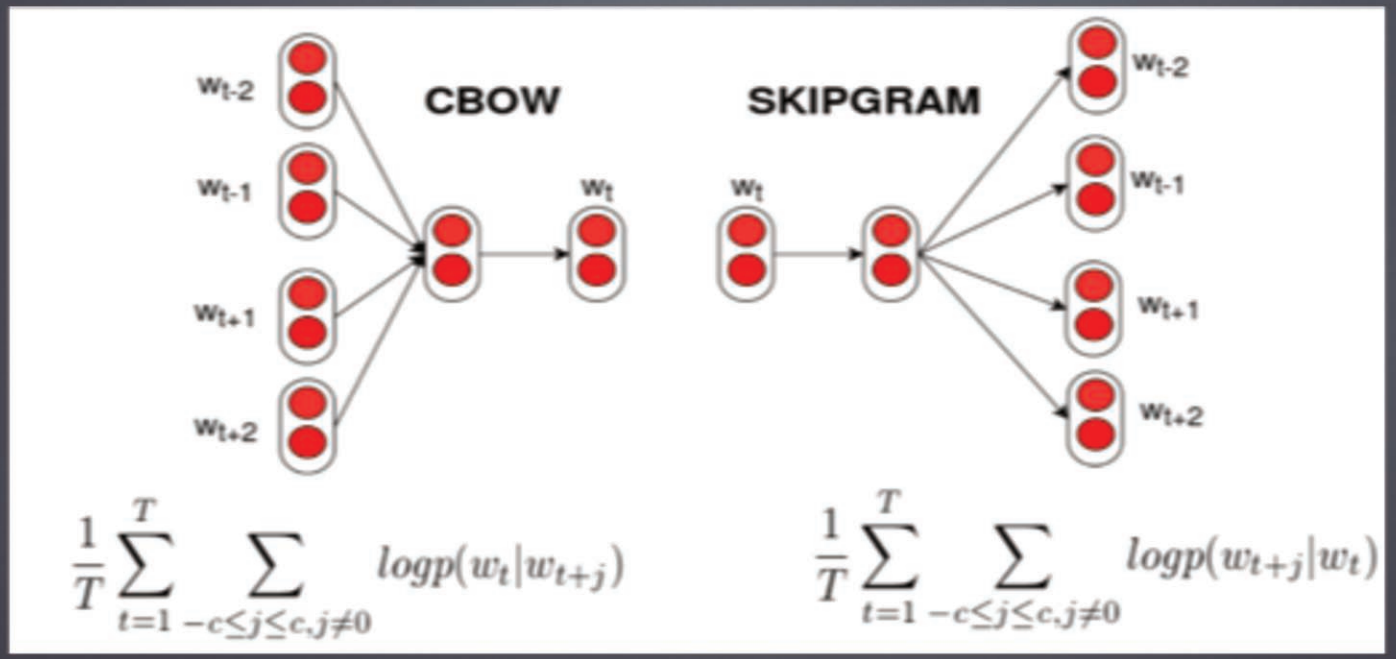
- As is customary in machine learning, we actually like to *minimize* things, so the cost function for word2vec is actually the negative log of the above
  - $J(\theta) = -\left(\frac{1}{T}\right) \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \neq 0}} \ln P\left(w_{t+j} \mid w_t ; \theta\right)$

Word2Vec

- $J(\theta) = -\left(\frac{1}{T}\right) \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \neq 0}} \ln P\left(w_{t+j} \mid w_t ; \theta\right)$
- But how do we calculate the conditional probabilities?
  - Surprisingly, we actually use *two* word vectors per word!
  - v_w When the word is a *center* word
  - u_w When the word is a *context* word
  - In the end, these will be used as our word embeddings!
- So consider some center word vector, v_c , and some context word vector, u_o .
  - $P(u_o|v_c) = \frac{\exp(u_o^T v_c)}{\sum_{i=1}^{|V|} \exp(u_i^T v_c)}$
- Then plug this into the loss function and minimize!

Word2Vec

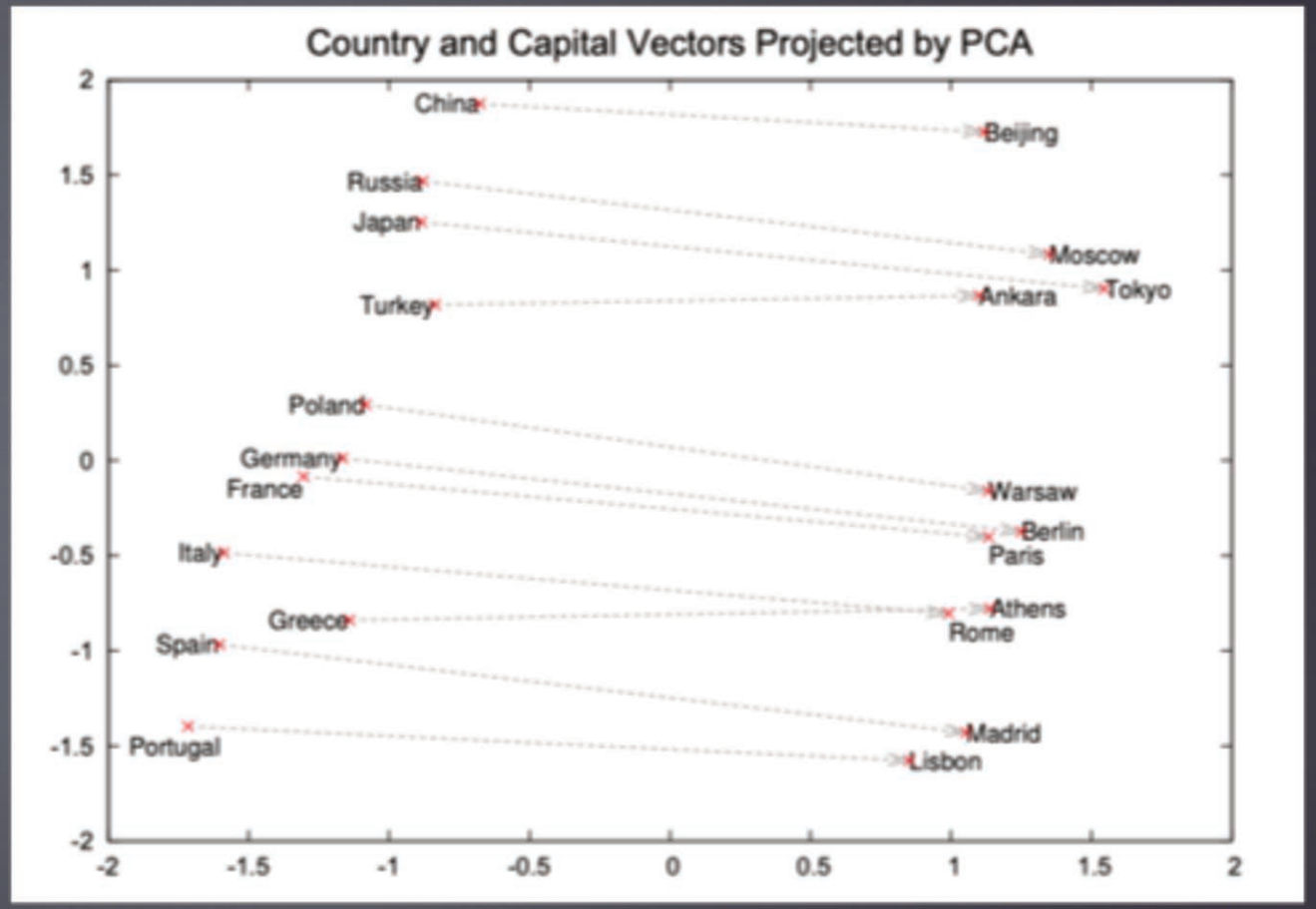
- In addition to the Skip-Gram version of word2vec, there is also the *Continuous Bag-Of-Words* (CBOW) architecture.
- Skip-Gram and CBOW, are fundamentally similar, the main difference between them being that we flip our classification target.



https://eu.wikipedia.org/wiki/Word2vec

Word2Vec – Results!

- Word2Vec showed a whole host of amazing results
- Not only does it succeed in capturing semantic similarity, but it captures *concepts*!



"Distributed Representations of Words and Phrases and their Compositionality"
https://arxiv.org/abs/1310.4546

Word2Vec – Results!

- We can add word2vec vectors together an get some amazing results!

Czech + currency	Vietnam + capital	German + airlines	Russian + river	French + actress
koruna	Hanoi	airline Lufthansa	Moscow	Juliette Binoche
Check crown	Ho Chi Minh City	carrier Lufthansa	Volga River	Vanessa Paradis
Polish zolty	Viet Nam	flag carrier Lufthansa	upriver	Charlotte Gainsbourg
CTK	Vietnamese	Lufthansa	Russia	Cecile De

"Distributed Representations of Words and Phrases and their Compositionality"
https://arxiv.org/abs/1310.4546

- How did word2vec compare to other state-of-the-art models at the time?

Model (training time)	Redmond	Havel	ninjutsu	graffiti	capitulate
Collobert (50d) (2 months)	conyers lubbock keene	plauen dzerzhinsky osterreich	reiki kohona karate	cheesecake gossip dioramas	abdicate accede rearm
Turian (200d) (few weeks)	McCarthy Alston Cousins	Jewell Arzu Ovitz	- - -	gunfire emotion impunity	- - -
Mnih (100d) (7 days)	Podhurst Harlang Agarwal	Pontiff Pinochet Rodionov	- - -	anaesthetics monkeys Jews	Mavericks planning hesitated
Skip-Phrase (1000d, 1 day)	Redmond Wash. Redmond Washington Microsoft	Vaclav Havel president Vaclav Havel Velvet Revolution	ninja martial arts swordsmanship	spray paint graffiti taggers	capitulation capitulated capitulating

"Distributed Representations of Words and Phrases and their Compositionality"
https://arxiv.org/abs/1310.4546

Word2Vec

- Summary
  - Word2Vec is a word embedding method used to create word vectors
  - It does so by training a language model
    - Training the language model is a sort of “dummy” task
    - We actual don’t care about the model itself. It’s the model’s parameters that we use as word vectors!
  - Common architectures are Skip-Gram and CBOW
    - Skip-Gram is typically used for larger corpora
    - CBOW tends to work better for smaller corpora
- Google provides pre-trained word2vec vectors that you can download and use!
- There are excellent NLP libraries in Python to train your own word2vec model!

GloVe

GloVe

- Word2Vec often works very well, but researchers quickly began looking into alternate methods
- The Global Vector model (Pennington, Socher, & Manning) provides an alternate way to create dense representations of words
- Rather than looking at context windows one at time, GloVe seeks to leverage global word co-occurrence statistics to produce word vectors

GloVe

- Let X be the global word co-occurrence matrix, where X_{ij} represents the number of times word j appears within the context of word i
- The total number of times word i appears within *any* context is then

$$X_i = \sum_j X_{ij}$$

- The relative probability of word j appearing in the context of word i is then

$$P(j \mid i) = \frac{X_{ij}}{X_i}$$

GloVe

Consider this tiny corpus with a context window size of 1:

- I like NLP
- Learning machine learning is great
- NLP is great

	I	Like	NLP	Learning	Machine	Is	Great
I	0	1	0	0	0	0	0
Like	1	0	1	0	0	0	0
NLP	0	1	0	0	0	1	0
Learning	0	0	0	0	2	1	0
Machine	0	0	0	2	0	0	0
Is	0	0	1	1	0	0	2
Great	0	0	0	0	0	2	0

GloVe

- One of the key insights of the GloVe model is that it is the *ratios* of co-occurrence probabilities that encodes meaning

	x = solid	x = gas	x = water	x = random
P(x | ice)	Large	Small	Large	Small
P(x | steam)	Small	Large	Large	Small
P(x | ice) / P(x | steam)	Very large	Very small	~ 1	~ 1

GloVe

- One of the key insights of the GloVe model is that it is the *ratios* of co-occurrence probabilities that encodes meaning

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

https://nlp.stanford.edu/pubs/glove.pdf

- Interesting idea! But how does the model actually work?

GloVe

- The starting point is to fit the GloVe vectors to the co-occurrence probabilities using a *log-bilinear* model

$$\widetilde{w_i}^T w_j = \log P_{ij} = \log X_{ij} - \log X_i$$

- As in word2vec, we again use center/context word vectors
- Why use a model of this form?
  - This is a simple way that makes the difference of word vectors proportional to the ratio of their co-occurrence probabilities!

$$\widetilde{w_x}^T (w_i - w_j) = \log P_{ix} - \log P_{jx} = \log \frac{P_{ix}}{P_{jx}}$$

- The take-away here is that by demanding a model of this form, we can encode meaning into the differences of word vectors

GloVe

- There's a little more to the story though!
- In GloVe, the difference between center word vectors and context word vectors is arbitrary, and so the model should be designed to respect this symmetry.
- That is, if we exchange *both*

$$w_i \leftrightarrow \widetilde{w_i}$$

$$X \leftrightarrow X^T$$

- Then our model should remain the same.

GloVe

- Let's see if this works! Recall the GloVe model so far,

$$\widetilde{w_i}^T w_j = \log X_{ij} - \log X_i$$

- Doing the exchange we get:

$$\widetilde{w_j}^T w_i = \log X_{ji} - \log X_i$$

- The exchange symmetry gets spoiled by the last term! We can fix this though by adding in a couple of bias vectors.

GloVe

- Adding in these bias terms, we get

$$\widetilde{w_i}^T w_j + b_j + \widetilde{b_i} = \log X_{ij}$$

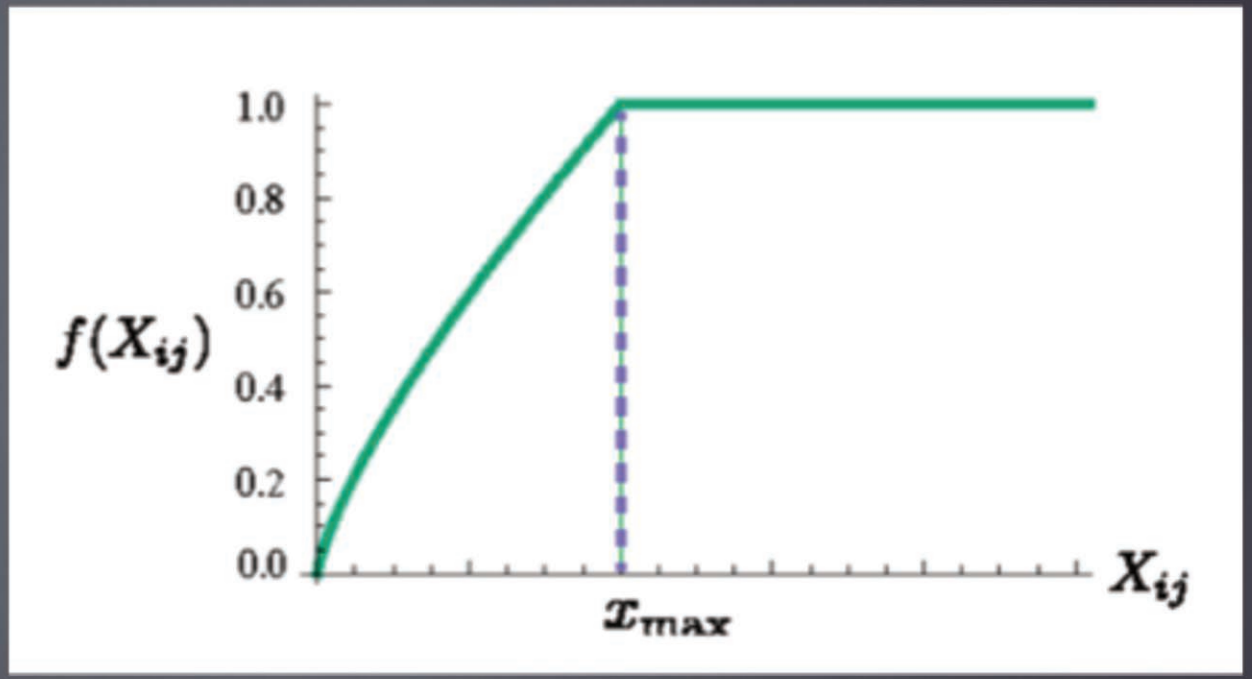
- Great! But there is one last pestering detail (I promise!). How do we handle stop words?

- GloVe introduces a weighting function to limit the effect of very frequent words.

GloVe

- The weighting function typically used in the GloVe model is

$$f(x) = \begin{cases} \left(\frac{x}{x_{max}}\right)^{\alpha}, & x < x_{max} \\ 1, & otherwise \end{cases}$$



https://nlp.stanford.edu/pubs/glove.pdf

GloVe

- Incorporating this weighting function into the model, we finally arrive at the loss function for GloVe!

$$J = \sum_{i,j=1}^{|V|} f(X_{ij})(\widetilde{w_i}^T w_j + b_j + \widetilde{b_i} - \log X_{ij})^2$$

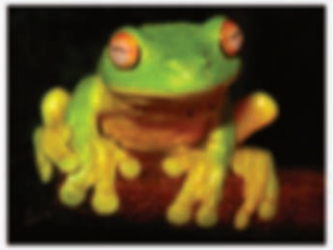
- This is a weighted least-squares problem!

GloVe – Results!

1. Nearest neighbors

The Euclidean distance (or cosine similarity) between two word vectors provides an effective method for measuring the linguistic or semantic similarity of the corresponding words. Sometimes, the nearest neighbors according to this metric reveal rare but relevant words that lie outside an average human's vocabulary. For example, here are the closest words to the target word *frog*:

0. *frog*
1. frogs
2. toad
3. litoria
4. leptodactylidae
5. rana
6. lizard
7. eleutherodactylus



3. litoria

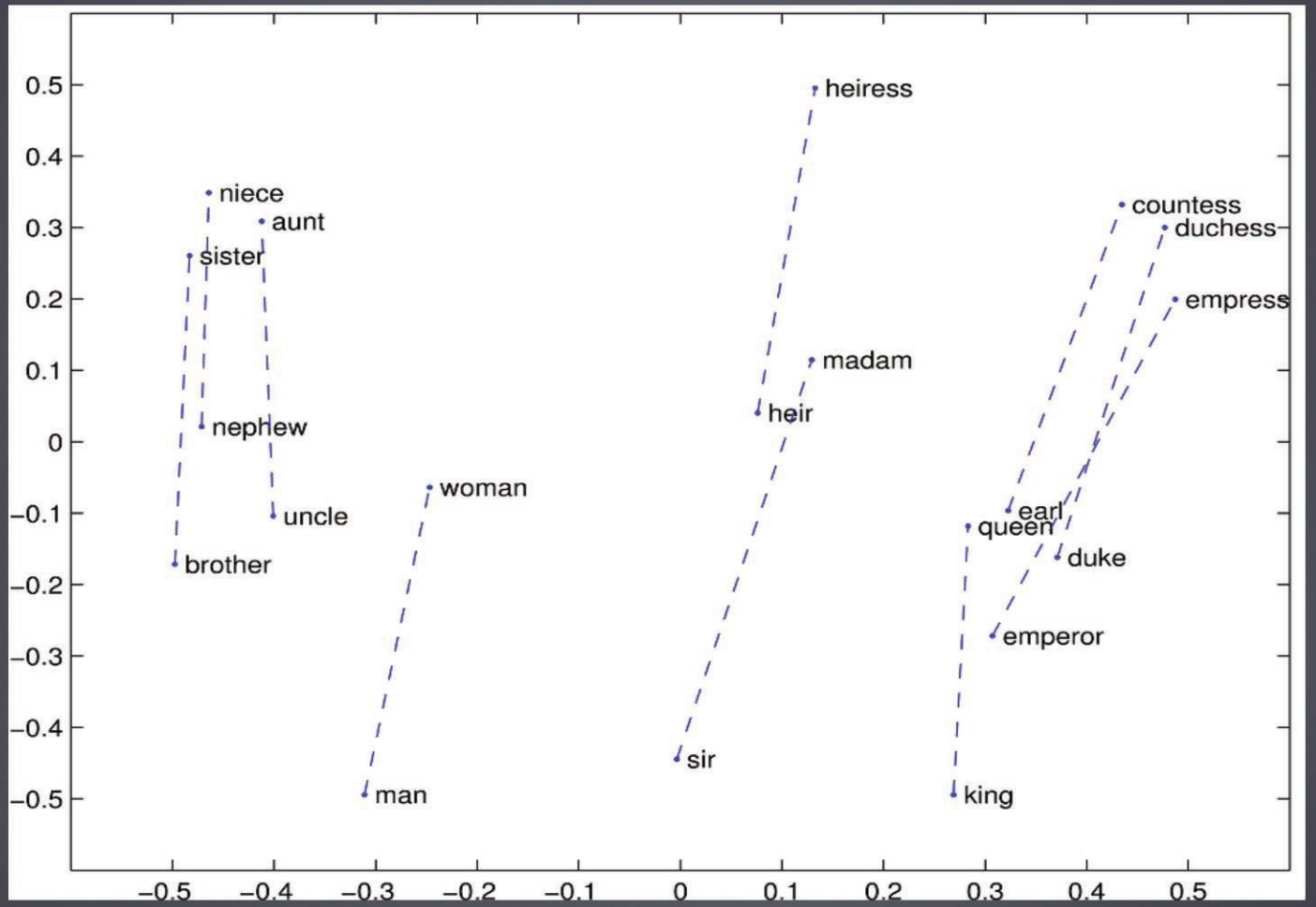
4. leptodactylidae

5. rana

7. eleutherodactylus

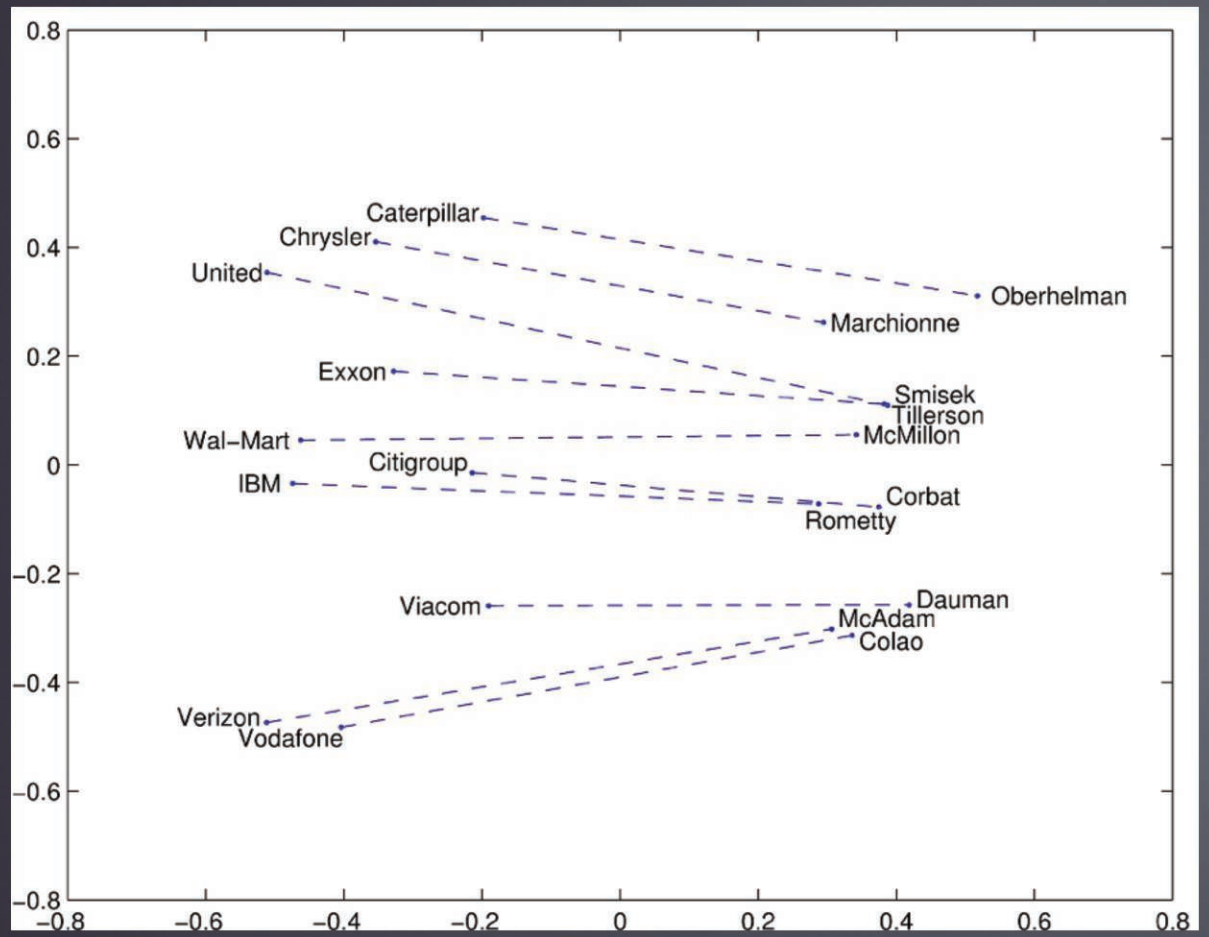
https://nlp.stanford.edu/projects/glove/

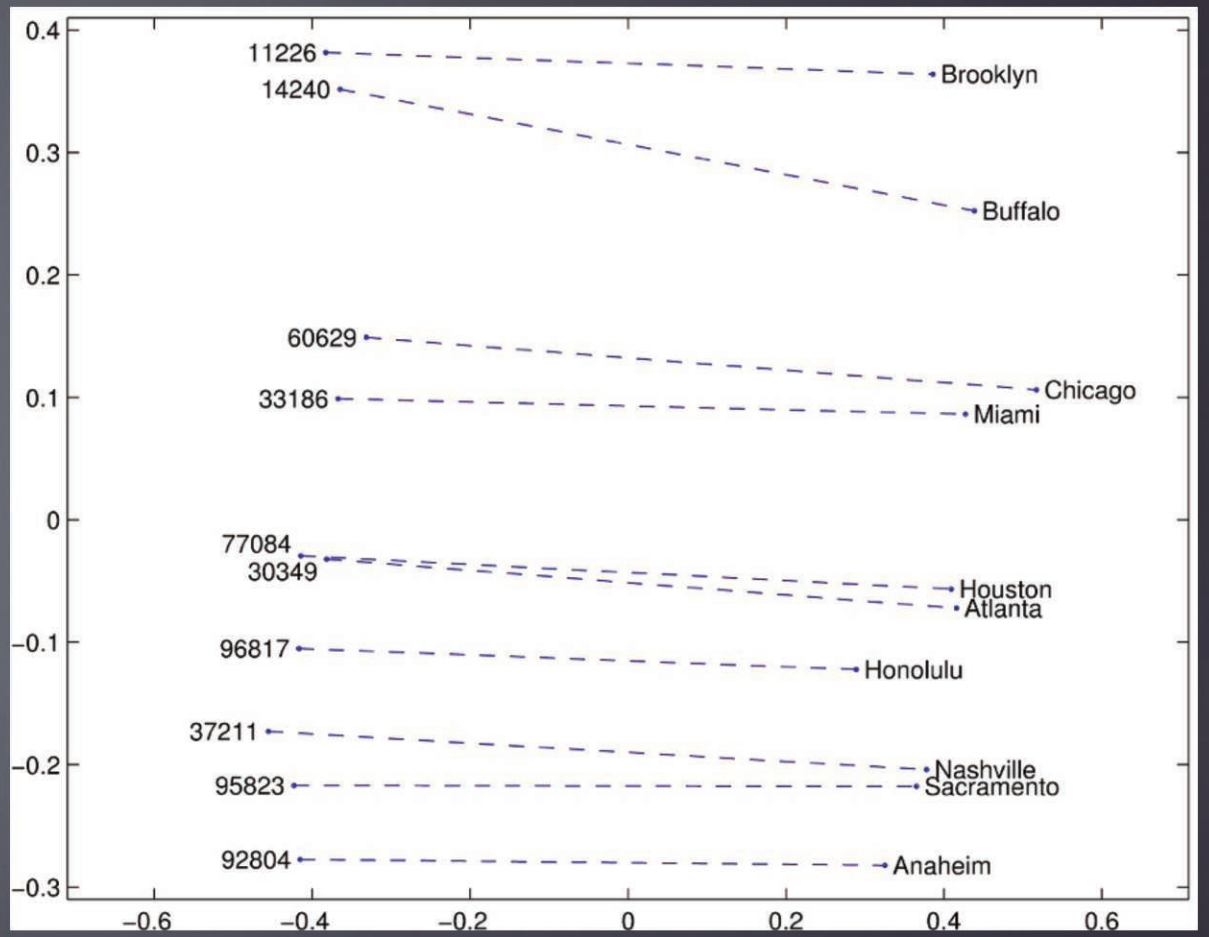
GloVe – Results!



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

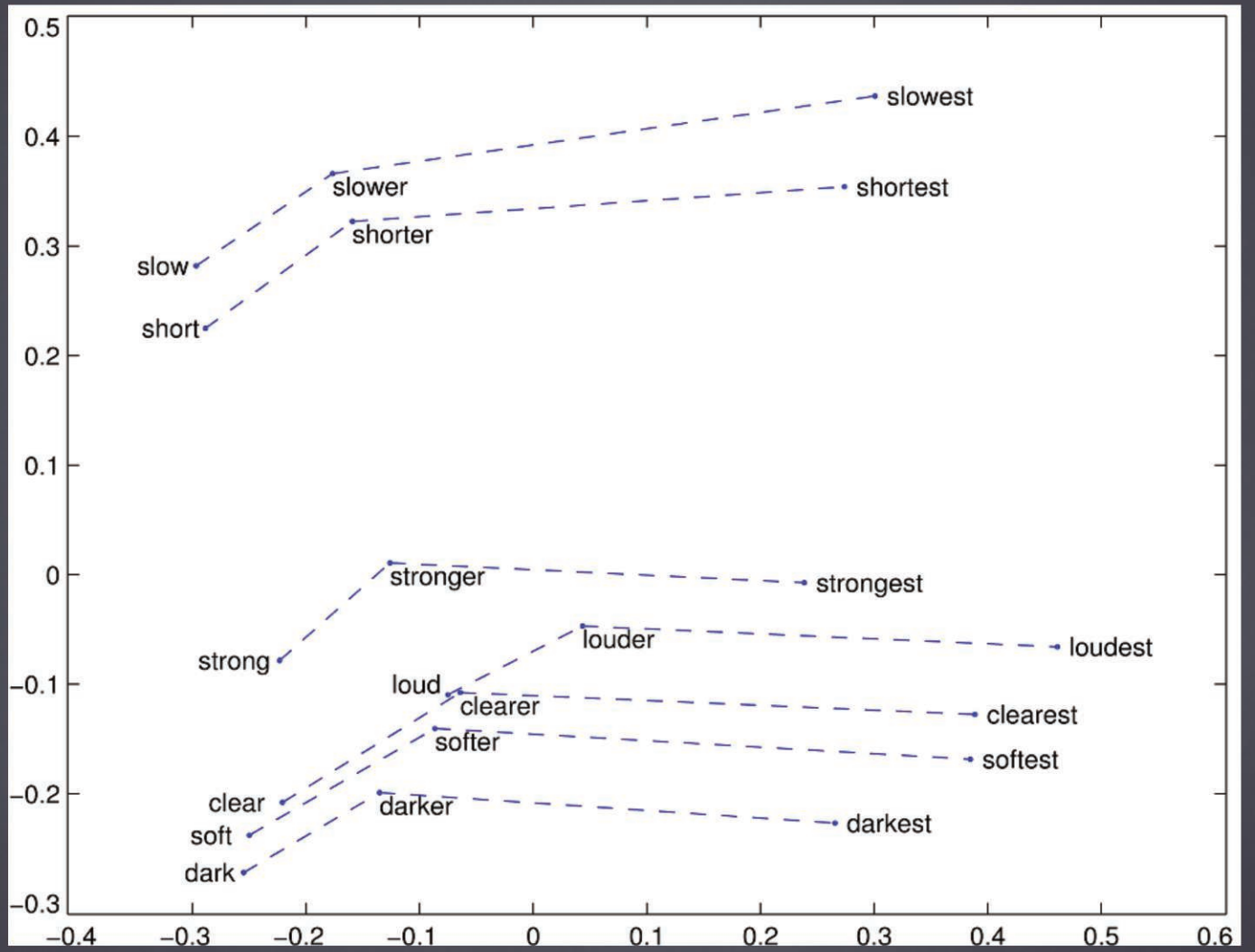
GloVe – Results!





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

GloVe – Results!



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

Evaluating Word Embeddings

Evaluating Word Embeddings

- What makes up a “good” word embedding?
- This is a tricky question! Word embeddings are typically used in service of some down-stream task (like classification).
- We can’t use performance on this downstream task to evaluate the word embeddings.
- To address this, Mikolov et al. (word2vec group) proposed analogy tasks!

Evaluating Word Embeddings

- The challenge is to use word embeddings to complete analogies.
- High performance on this task typically means the word embeddings better capture the “essence” of words.

a : b :: c : ?

King : Queen :: Man : Woman (Semantic Analogy)

Dance : Dancing. :: Swim : Swimming (Syntactic Analogy)

Evaluating Word Embeddings

- How does this look mathematically?
- Given the analogy task (a, b, c are given, word d is unknown):
  - a : b :: c : d
- Our guess for word d, is whatever word has the largest *cosine similarity*

$$d = \max_x \frac{(w_b - w_a + w_c)^T w_x}{\|w_b - w_a + w_c\| \, \|w_x\|}$$

Evaluating Word Embeddings

- How do various word embedding methods compare?

Model	Dim.	Size	Sem.	Syn.	Tot.
ivLBL	100	1.5B	55.9	50.1	53.2
HPCA	100	1.6B	4.2	16.4	10.8
GloVe	100	1.6B	67.5	54.3	60.3
SG	300	1B	61	61	61
CBOW	300	1.6B	16.1	52.6	36.1
vLBL	300	1.5B	54.2	64.8	60.0
ivLBL	300	1.5B	65.2	63.0	64.0
GloVe	300	1.6B	80.8	61.5	70.3
SVD	300	6B	6.3	8.1	7.3
SVD-S	300	6B	36.7	46.6	42.1
SVD-L	300	6B	56.6	63.0	60.1
CBOW†	300	6B	63.6	67.4	65.7
SG†	300	6B	73.0	66.0	69.1
GloVe	300	6B	77.4	67.0	71.7
CBOW	1000	6B	57.3	68.9	63.7
SG	1000	6B	66.1	65.1	65.6
SVD-L	300	42B	38.4	58.2	49.2
GloVe	300	42B	**81.9**	**69.3**	**75.0**

https://nlp.stanford.edu/pubs/glove.pdf

Beyond GloVe and Word2Vec

Beyond GloVe and Word2Vec

- Word2Vec and GloVe are both excellent, but no model is ever perfect.

- The word vectors produced by these models are static, and may have difficulty with word-sense disambiguation.
  - Does “bank” refer to the place I keep my money, or to a landscape feature near a river?

- More recently, deep-learning methods have been used to create *contextualized word embeddings* to deal with this problem. Some notable examples:
  - ELMo
  - ULMFiT
  - OpenAI Transformer
  - BERT

Summary

- Word embeddings solve of the problem of representing words/documents in a way that we can feed into other Data Science and Machine Learning tasks
- Count Vectorization and TF-IDF vectors provide more intuitive methods for generating word embeddings
  - Using these methods along with bigrams/trigrams can yield pretty good results!
- Word2Vec and GloVe are more advanced methods of creating dense representations of word vectors
  - The math of this might get a little sticky, but there are plenty of Python resources to handle things under-the-hood.
- Performance on analogy tasks is the usual way we evaluate the efficacy of a word embedding

Further Reading

- Great Blog Post on Word2Vec Intuition: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/
- GloVe Paper: https://nlp.stanford.edu/pubs/glove.pdf
- Stanford Lecture notes on word vectors: http://web.stanford.edu/class/cs224n/lectures/lecture2.pdf
- TensorFlow tutorial on word vectors: https://www.tensorflow.org/tutorials/representation/word2vec