Salvador Medina Herrera
(with slides from José A. R. Fonollosa and Marta R. Costa-Jussà)

WORD VECTORS

Outline

- **Motivation**
  - One-hot Encoding
  - Vectors and Documents
  - TF-IDF Vectors
  - PPMI Vectors
- Types of Word Vectors
  - Knowledge-based
  - Corpus-based
  - Word2Vec (CBOW)
  - Word2Vec (Skip-Gram)
  - Others (FastText, Char-based,…)
- Visualization and Evaluation

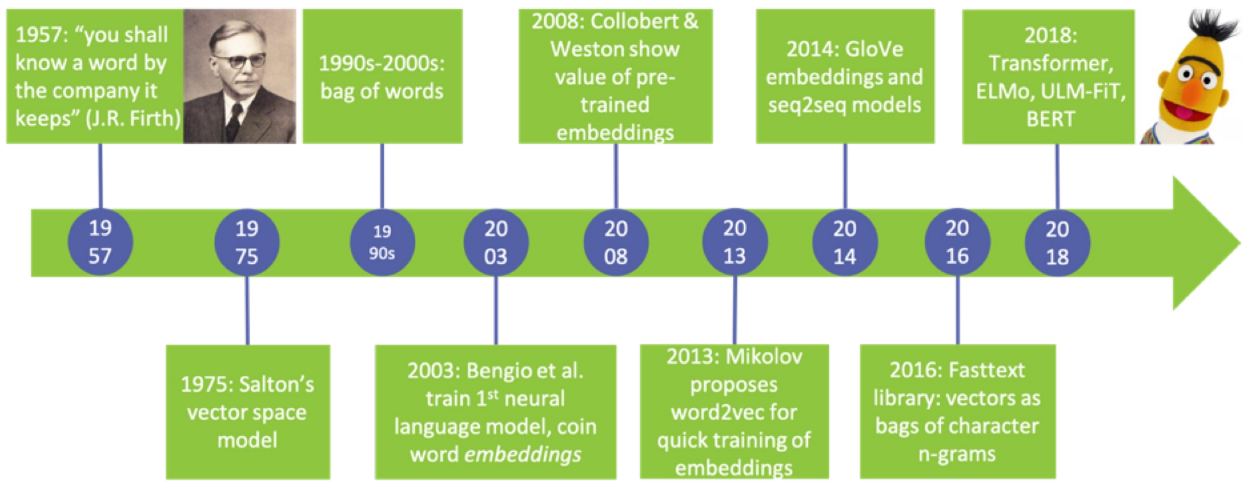
Question

- What do you know about Word Vectors or Word Embeddings?

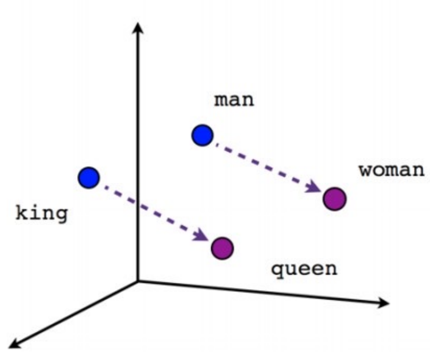
A Word embedding is a numerical representation of a word

- Word embeddings allow for arithmetic operations on a text
  - Example: time + flies
- Word embeddings have been refered also as:
  - Semantic Representation of Words
  - Word Vector Representation

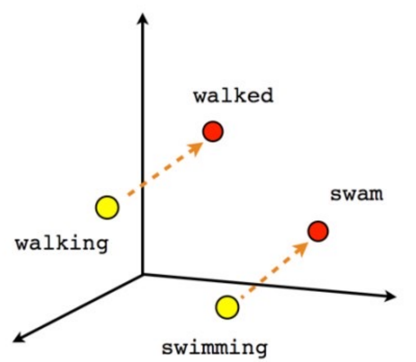
Timeline


1957: “you shall know a word by the company it keeps” (J.R. Firth)
1990s-2000s: bag of words
2008: Collobert & Weston show value of pre-trained embeddings
2014: GloVe embeddings and seq2seq models
2018: Transformer, ELMo, ULM-FiT, BERT
19 57
19 75
19 90s
20 03
20 08
20 13
20 14
20 16
20 18
1975: Salton’s vector space model
2003: Bengio et al. train 1st neural language model, coin word *embeddings*
2013: Mikolov proposes word2vec for quick training of embeddings
2016: Fasttext library: vectors as bags of character n-grams

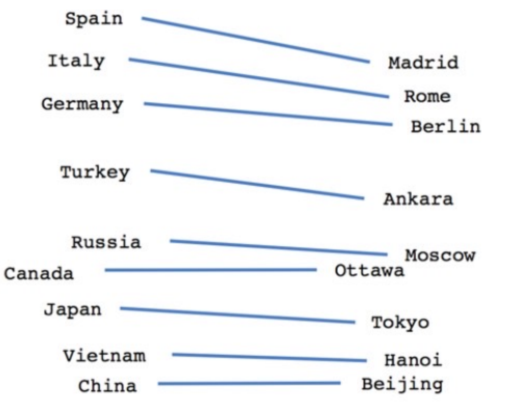
Word vectors



Male-Female



Verb tense



Country-Capital

Distributional Hypothesis Contextuality

- Never ask for the meaning of a word in isolation, but only in the context of a sentence
  (Frege, 1884)

- For a large class of cases... the meaning of a word is its use in the language
  (Wittgenstein, 1953)

- You shall know a word by the company it keeps (Firth, 1957)

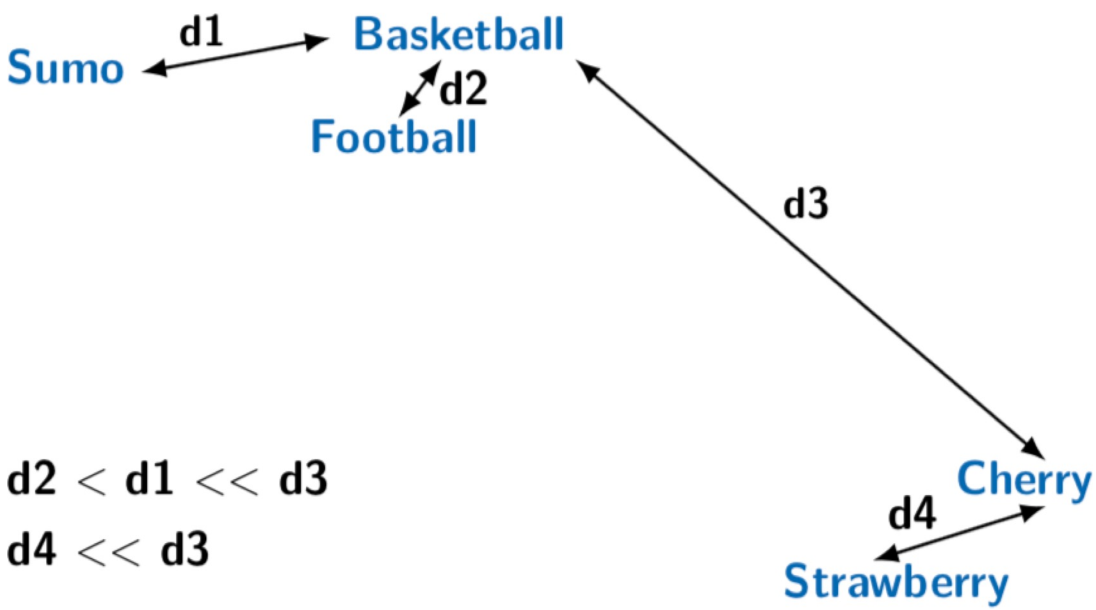
- Words that occur in similar contexts tend to have similar meaning
  (Harris, 1954)

Words embeddings allow to process sentences with ML

- Sentences are sequences of symbols
- Word vectors (word embeddings) are vector representations of words, the “natural” unit for solving natural language processing tasks.

id	qid1	qid2	question1	question2	is_duplicate
447	895	896	What are natural numbers?	What is a least natural number?	0
1518	3037	3038	Which pizzas are the most popularly ordered pizzas on Domino's menu?	How many calories does a Dominos pizza have?	0
3272	6542	6543	How do you start a bakery?	How can one start a bakery business?	1
3362	6722	6723	Should I learn python or Java first?	If I had to choose between learning Java and Python, what should I choose to learn first?	1

Vector representations can help us finding similar meanings ...need for a concept of distance to be defined.



d2 < d1 << d3

d4 << d3

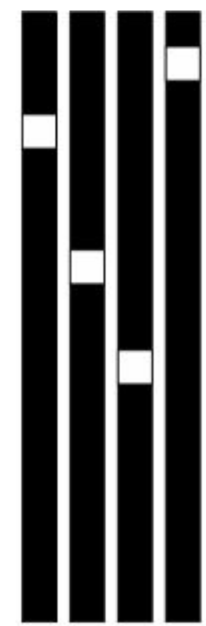
Outline

- Motivation
  - **One-hot Encoding**
  - Vectors and Documents
  - TF-IDF Vectors
- Types of Word Vectors
  - Knowledge-based
  - Corpus-based
  - PPMI Vectors
  - Word2Vec (CBOW)
  - Word2Vec (Skip-Gram)
  - Others (FastText, Char-based,…)
- Visualization and Evaluation

How to represent a word: One-hot vectors

- **One hot vector** (dim == vocabulary size)
  - Very large vector (millions of words in some applications)
  - Sparse, orthogonal representations
  - No information about how words are related
  - No useful vector distance
  - Huge use of memory (if sparse matrices are not used)
  - Usual coding of categorical variables for Linear models and SVMs with the standard kernels

```
[ [0 1 0 0 0 0 ...]   # to    (1)
  [0 0 0 1 0 0 ...]   # be    (3)
  [0 0 1 0 0 0 ...]   # or    (2)
  [0 0 0 0 0 1 ...]   # not   (5)
  [0 1 0 0 0 0 ...]   # to    (1)
  [0 0 0 1 0 0 ...]   # be    (3)
```



Outline

- Motivation
  - One-hot Encoding
  - **Vectors and Documents**
  - TF-IDF Vectors
- Types of Word Vectors
  - Knowledge-based
  - Corpus-based
  - PPMI Vectors
  - Word2Vec (CBOW)
  - Word2Vec (Skip-Gram)
  - Others (FastText, Char-based,…)
- Visualization and Evaluation

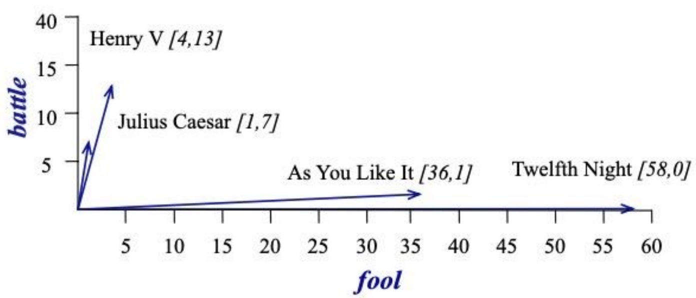
Vectors and Documents

- **term-document matrix**: number of times a term (row) appears in a document (column)
- Originally defined as a means of finding similar documents for the task of document information retrieval
- We can use document vectors to find other similar documents

Document vectors

	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

Term-document matrix for four words (rows) in four Shakespeare plays.



The comedies have high values for the fool dimension and low values for the battle dimension.

Vectors and Documents

- **term-document matrix**: number of times a term (row) appears in a document (column)
- Similar words have similar vectors because they tend to occur in similar documents

	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

❌ Problems:

- Hard to get meaningful results for frequent words (the, it...)
- 'good' appears frequently in different contexts

✅ Solution:

- tf-idf

Outline

- Motivation
  - One-hot Encoding
  - Vectors and Documents
  - **TF-IDF Vectors**
- Types of Word Vectors
  - Knowledge-based
  - Corpus-based
  - PPMI Vectors
  - Word2Vec (CBOW)
  - Word2Vec (Skip-Gram)
  - Others (FastText, Char-based,…)
- Visualization and Evaluation

TF-IDF

- **t**erm **f**requency - **i**nverse **d**ocument **f**requency
- Used when the dimensions are documents

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

Term frequency (tf)

- Number of times a term occurs in a document

$$\mathrm{tf}_{t,d} = \mathrm{count}(t,d)$$

$$\mathrm{tf}_{t,d} = \log_{10}(\mathrm{count}(t,d)+1)$$

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

Document frequency (df)

- Number of documents a term occurs in
- Higher weight to words that occur in few documents

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

TF-IDF

	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

After tf-idf weighting

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.074	0	0.22	0.28
good	0	0	0	0
fool	0.019	0.021	0.0036	0.0083
wit	0.049	0.044	0.018	0.022

Outline

- Motivation
  - One-hot Encoding
  - Vectors and Documents
  - TF-IDF Vectors
- **Types of Word Vectors**
  - Knowledge-based
  - Corpus-based
  - PPMI Vectors
  - Word2Vec (CBOW)
  - Word2Vec (Skip-Gram)
  - Others (FastText, Char-based,…)
- Visualization and Evaluation

Beyond one-hot: Type of word vectors

I. Based on human knowledge
II. Based on context words:
"You shall know a word by the company it keeps" (J. R. Firth 1957)

I will go to the **cinema** on Sunday.
Pop-up **cinema** to enjoy films about local cuisine.
Concerning eyesight, photography, **cinema**, television.
I will go to your **office** on Tuesday.

1. – Count-based methods (co-occurrence counts)
2. – Direct prediction / Deep learning methods
3. – Hybrid, (GloVe vectors)

Outline

- Motivation
  - One-hot Encoding
  - Vectors and Documents
  - TF-IDF Vectors
- Types of Word Vectors
  - **Knowledge-based**
  - Corpus-based
  - PPMI Vectors
  - Word2Vec (CBOW)
  - Word2Vec (Skip-Gram)
  - Others (FastText, Char-based,…)
- Visualization and Evaluation

Word vectors based on human knowledge

Based on human-created linguistic resources, e.g. Wordnet, a thesaurus containing lists of **synonym sets** and **hypernyms** (“is a” relationships).

e.g. synonym sets containing “good”:

```
from nltk.corpus import wordnet as wn
poses = { 'n':'noun', 'v':'verb', 's':'adj (s)', 'a':'adj', 'r':'adv'}
for synset in wn.synsets("good"):
    print("{}: {}".format(poses[synset.pos()],
          ", ".join([l.name() for l in synset.lemmas()])))
```

```
noun: good
noun: good, goodness
noun: good, goodness
noun: commodity, trade_good, good
adj: good
adj (sat): full, good
adj: good
adj (sat): estimable, good, honorable, respectable
adj (sat): beneficial, good
adj (sat): good
adj (sat): good, just, upright
…
adverb: well, good
adverb: thoroughly, soundly, good
```

e.g. hypernyms of “panda”:

```
from nltk.corpus import wordnet as wn
panda = wn.synset("panda.n.01")
hyper = lambda s: s.hypernyms()
list(panda.closure(hyper))
```

```
[Synset('procyonid.n.01'),
Synset('carnivore.n.01'),
Synset('placental.n.01'),
Synset('mammal.n.01'),
Synset('vertebrate.n.01'),
Synset('chordate.n.01'),
Synset('animal.n.01'),
Synset('organism.n.01'),
Synset('living_thing.n.01'),
Synset('whole.n.02'),
Synset('object.n.01'),
Synset('physical_entity.n.01'),
Synset('entity.n.01')]
```

Question

- What problems can you imagine with this approach?

Word vectors based on human knowledge

Problems:

- There is no straightforward way to compute the similarity between words (to create a word vector)
- Missing nuance: binary relationship (e.g., synonyms only in some contexts)
- Limited number of words
- Impossible to keep up-to-date
- Subjective
- Costly human labor to create and adapt
- But can be used to complement other vector representations

Outline

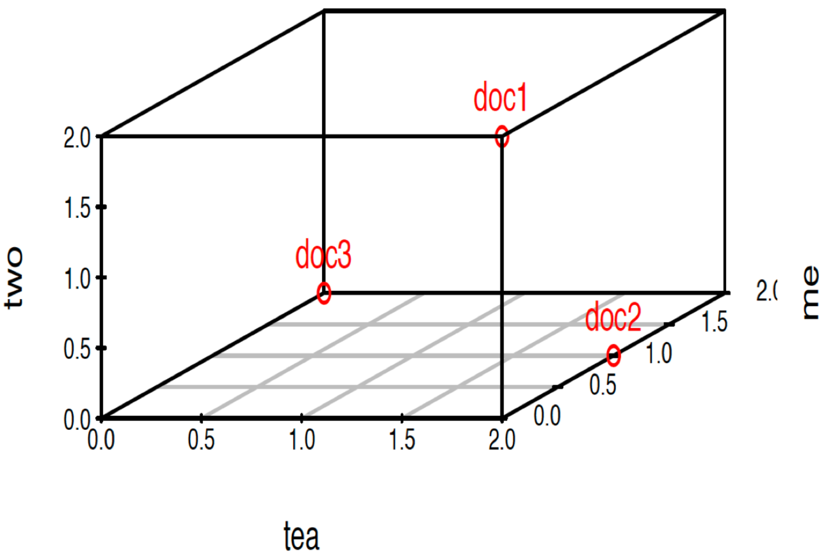
- Motivation
  - One-hot Encoding
  - Vectors and Documents
  - TF-IDF Vectors
- Types of Word Vectors
  - Knowledge-based
  - **Corpus-based**
  - PPMI Vectors
  - Word2Vec (CBOW)
  - Word2Vec (Skip-Gram)
  - Others (FastText, Char-based,…)
- Visualization and Evaluation

Based on context words: count-methods

- How we do this? What we need is a collection of documents, and using this documents, we can use different methods...
- Starting by **term-frequency**... counting the number of words appear in a document

doc1	Two for tea and tea for two
doc2	Tea for me and tea for you
doc3	You for me and me for you

	two	tea	me	you
doc1	2	2	0	0
doc2	0	2	1	1
doc3	0	0	2	2



Based on context words II.1

- II.1. Count-based + SVD (reduced rank aprox.)
  - Count word cooccurrence counts: two options
    - Window-base Word / Word cooccurrence matrix

Word-Word Matrix

Context: ± 7 words

sugar, a sliced lemon, a tablespoonful of	**apricot**	preserve or jam, a pinch each of,
their enjoyment. Cautiously she sampled her first	**pineapple**	and another fruit whose taste she likened
well suited to programming on the digital	**computer**.	In finding the optimal R-stage policy from
for the purpose of gathering data and	**information**	necessary for the study authorized in the

Resulting word-word matrix:

	aardvark	computer	data	pinch	result	sugar	...
apricot	0	0	0	1	0	1	
pineapple	0	0	0	1	0	1	
digital	0	2	1	0	1	0	
information	0	1	6	0	4	0	

Outline

- Motivation
  - One-hot Encoding
  - Vectors and Documents
  - TF-IDF Vectors
- Types of Word Vectors
  - Knowledge-based
  - Corpus-based
  - PPMI Vectors
  - Word2Vec (CBOW)
  - Word2Vec (Skip-Gram)
  - Others (FastText, Char-based,…)
- Visualization and Evaluation

Based on context words II.1

- II.1. Count-based + SVD (reduced rank aprox.)
  - Count word cooccurrence counts: two options
    - Window-base Word / Word cooccurrence matrix
    - Pointwise mutual information:
    - Do words x and y co-occur more than if they were independent?

$$\text{PMI}(word_1, word_2) = \log_2 \frac{P(word_1, word_2)}{P(word_1)P(word_2)}$$

Based on context words II.1

- II.1. Count-based + SVD (reduced rank aprox.)
  - Count word cooccurrence counts: two options
    - Window-base Word / Word cooccurrence matrix
    - Pointwise mutual information:

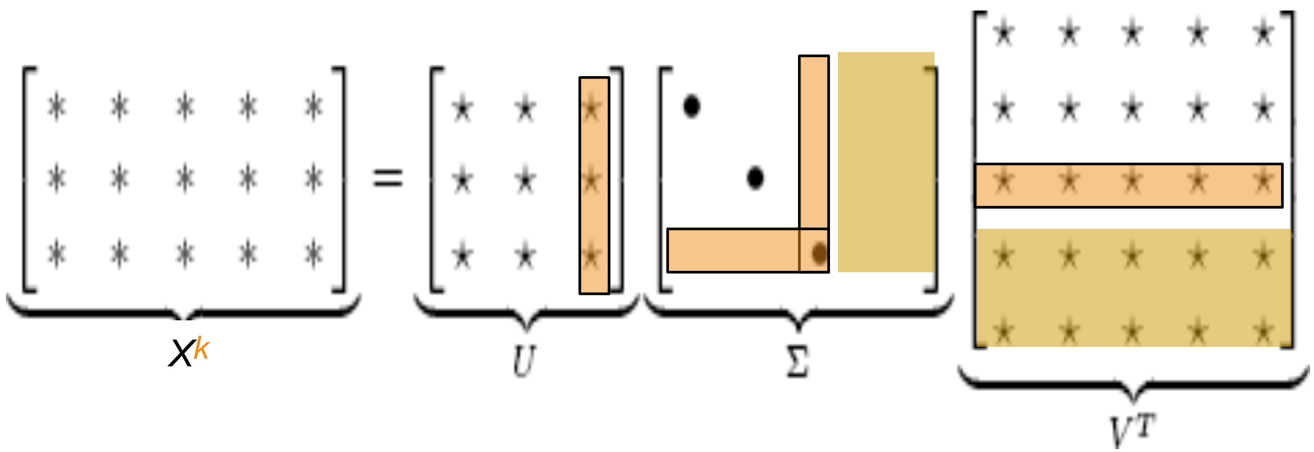
	aardvark	computer	data	pinch	result	sugar
apricot	0	0	0	1	0	1
pineapple	0	0	0	1	0	1
digital	0	2	1	0	1	0
information	0	1	6	0	4	0

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

$$pmi_{ij} = \log_2 \frac{p_{ij}}{p_{i*}p_{*j}} \qquad ppmi_{ij} = \begin{cases} pmi_{ij} & \text{if } pmi_{ij} > 0 \\ 0 & \text{otherwise} \end{cases}$$

Based on context words II.1

- II.1. Count-based + SVD (reduced rank aprox.)
  - Count word cooccurrence counts: two options
    - Word / documents cooccurrence matrix
    - Window-base Word / Word cooccurrence matrix
  - Singular Value Decomposition X = U S V^T to reduce the dimensionality (rank). The rows of U are the word embeddings.



Based on context words II.1

- II.1. Count-based + SVD (reduced rank aprox.)
  - Count word cooccurrence counts: two options
    - Word / documents cooccurrence matrix
    - Window-base Word / Word cooccurrence matrix
  - Singular Value Decomposition $X = U S V^T$ to reduce the dimensionality (rank). The rows of U are the word embeddings.
- Problems
  - Function words (the, you, is, ..) have a big impact
    - Solutions: modify raw counts (log, tf-idf) or remove function words.
  - High dimensional matrix.
  - Quadratic cost of SVD
    - Solutions: adaptive algorithms

Outline

- Motivation
  - One-hot Encoding
  - Vectors and Documents
  - TF-IDF Vectors
  - PPMI Vectors
- Types of Word Vectors
  - Knowledge-based
  - **Corpus-based**
  - Word2Vec (CBOW)
  - Word2Vec (Skip-Gram)
  - Others (FastText, Char-based,…)
- Visualization and Evaluation

Based on context words: direct prediction

- Continuous space representations or word embeddings
- Small vector of real numbers (dim 200 – 400)
- Linguistic or semantic similarity can be measured with the Euclidean distance or cosine similarity.
- Vector differences capture word relations
- Standard choice for deep learning models

(12424, 100)

	0	1	2	3	4	5	6	7	8	9	...	90	91	92	93
shall	-0.002272	0.015870	0.018349	0.022802	0.028364	-0.040064	-0.013263	0.136607	0.019667	0.033407	...	0.037663	-0.087140	0.073169	-0.028257
unto	0.034425	-0.102070	0.018051	0.017960	0.172954	-0.115672	-0.012632	0.096919	-0.049203	-0.040344	...	0.106373	-0.075703	0.013888	-0.134224
lord	0.051990	-0.113865	0.007226	0.031754	0.052963	-0.094523	-0.067664	0.001706	-0.112827	-0.078586	...	-0.041636	0.053685	0.041299	-0.026255
thou	-0.152183	-0.073681	-0.091472	0.022033	0.008415	-0.048438	-0.041181	0.082019	0.004648	0.044870	...	0.101531	-0.018404	-0.070462	-0.041363
thy	-0.257579	-0.023008	0.053303	0.013690	-0.083293	0.034279	0.078811	0.079851	-0.015215	-0.111211	...	-0.064527	0.112085	0.061625	0.026398

5 rows × 100 columns

Outline

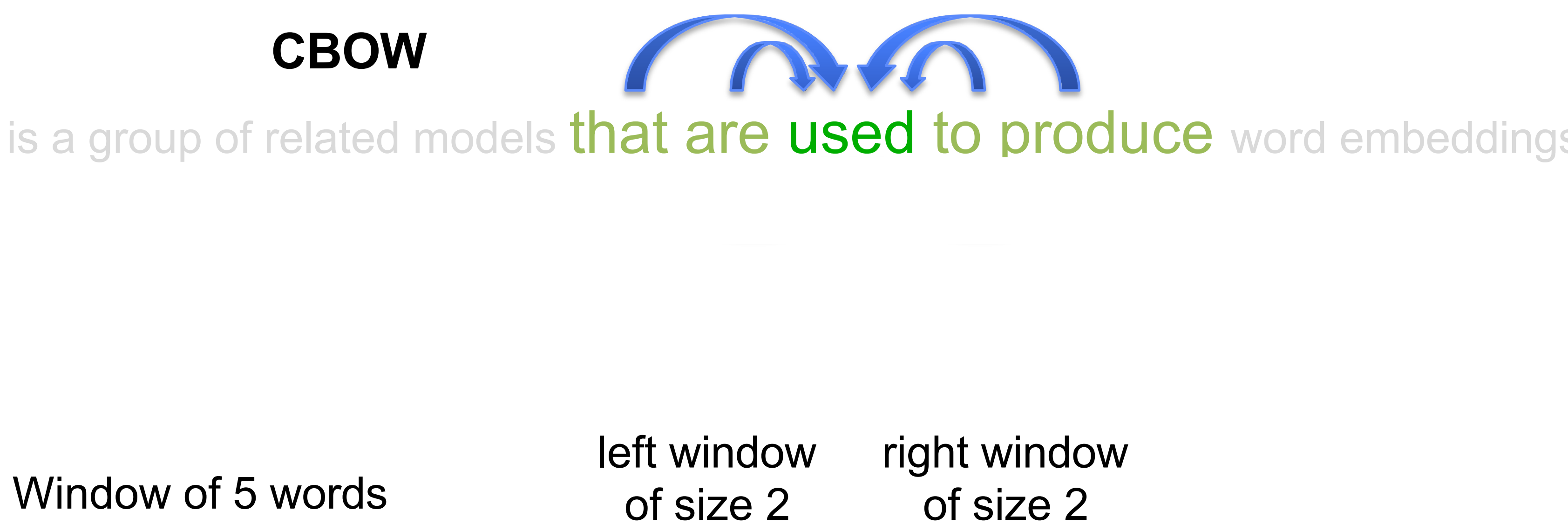
- Motivation
  - One-hot Encoding
  - Vectors and Documents
  - TF-IDF Vectors
  - PPMI Vectors
- Types of Word Vectors
  - Knowledge-based
  - Corpus-based
  - **Word2Vec (CBOW)**
  - Word2Vec (Skip-Gram)
  - Others (FastText, Char-based,…)
- Visualization and Evaluation

Based on context words II.2a

II.2 Direct prediction / Deep learning methods

Word2vec (Mikolov, Google 2013) two models:

- CBOW (Continuous bag-of-words): prediction of a word using the context words (bag-of-words)



Continuous bag-of-words (CBOW)

FUN WITH FILL-INS

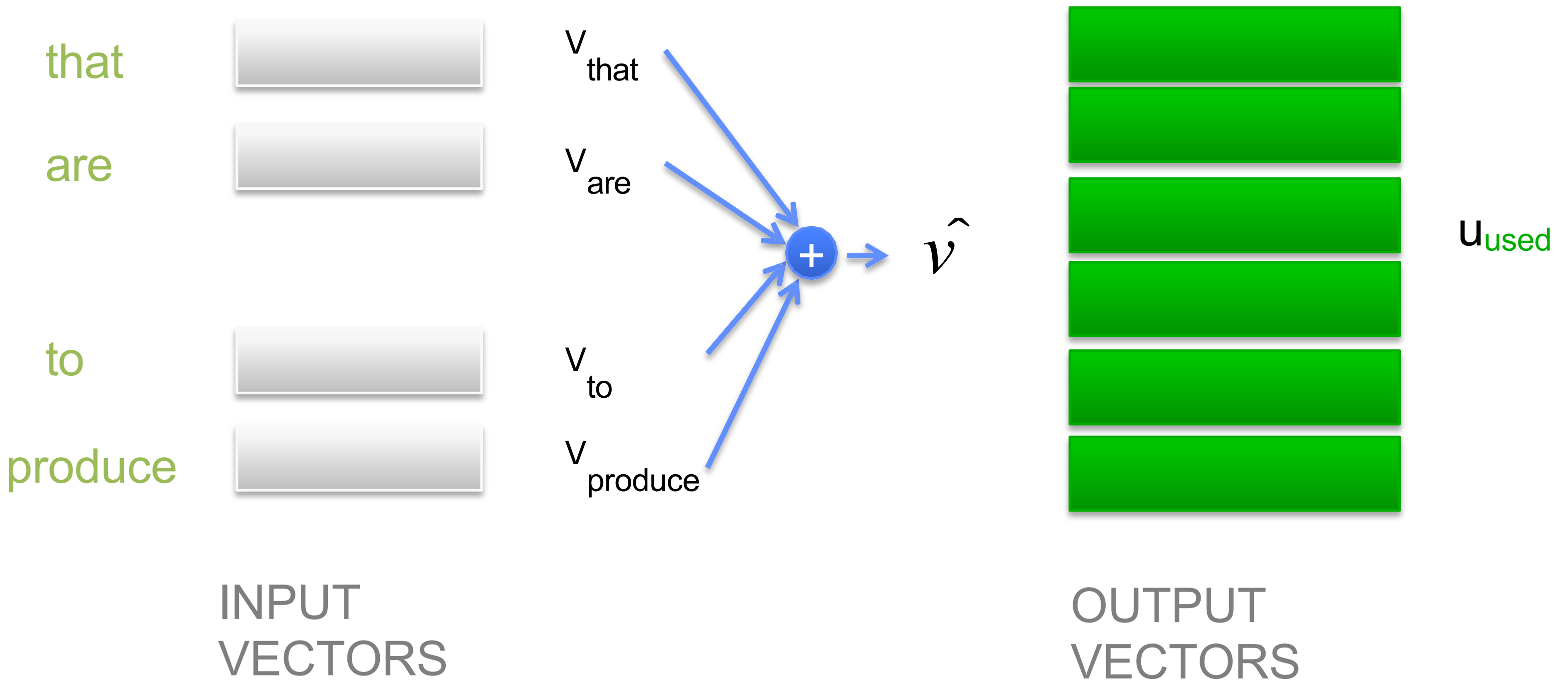
First Grade Sight Words

Choose the sight word from the Word List that will complete each sentence below.
Hint: Words can be used more than once.

Word List: are, good, now

1. Plums ________ in a tree.
2. Are the plums ________ now?
3. The plums are hard. They ________ not good.
4. Sun is good for plums. Rain is ________ for plums.
5. Are the plums good __________?
6. The plums ________ soft.
7. ________ the plums are good!

Based on context words II.2a



CBOW

CBOW equations

Continuous bag-of-words (CBOW)
W is the word vocabulary
Input vectors: v_w for each w 2 W
Output vectors: u_w for each w 2 W
The 'predicted' output word vector is the sum over all context input vectors:

$$\hat{v} = \sum_{c-m \leq i \leq c+m, i \neq c} v_{w_i} = \sum_{c-m \leq i \leq c+m, i \neq c} v_i$$

We use the dot product to compute the score vector (word similarity):

$$z_w = u_w^T \hat{v}$$

And the softmax function to get probabilities

$$p(w_c | w_{c-m} \cdots w_{c-1}, w_{c+1} \cdots w_{c+m}) = \frac{exp(z_c)}{\sum_{w \in W} exp(z_w)}$$

$$\frac{exp(u_c^T \hat{v})}{\sum_{w \in W} exp(u_w^T \hat{v})}$$

CBOW equations

The standard choice for the loss function is the cross-entropy of the estimated probability $p(w)$ respect to the true probability $q(w)$

$$
\begin{aligned}
CE(q,p) &= E_q[-\log p(w)] \\
&= E_q[-\log p(w) + \log q(w) - \log q(w)] \\
&= E_q[\frac{\log q(w)}{\log p(w)}] + E_q[-\log q(w)] \\
&= D_{KL}(q||p) + H(q)
\end{aligned}
$$

that in our case is equivalent to the minimization of the negative log-likelihood of the target word vector given the context

$$
\begin{aligned}
J &= -\sum_c \log p(w_c | w_{c-m} \cdots w_{c-1}, w_{c+1} \cdots w_{c+m}) \\
&= -\sum_c \log \frac{\exp(u_c^T \hat{v})}{\sum_{w \in W} \exp(u_w^T \hat{v})} \\
&= -\sum_c u_c^T \hat{v} + \log \sum_{w \in W} \exp(u_w^T \hat{v})
\end{aligned}
$$

Outline

- Motivation
  - One-hot Encoding
  - Vectors and Documents
  - TF-IDF Vectors
  - PPMI Vectors
- Types of Word Vectors
  - Knowledge-based
  - Corpus-based
  - Word2Vec (CBOW)
  - **Word2Vec (Skip-Gram)**
  - Others (FastText, Char-based,…)
- Visualization and Evaluation

Based on context words II.2a

II.2 Direct **prediction** / Deep learning methods

Word2vec (Mikolov, Google 2013) two models:

- Continuous skip-gram architecture: prediction of the context words using the current word

is a group of related models that are used to produce word embeddings

skip-gram

Window of 5 words

left window
of size 2

right window
of size 2

Step-by-step: skip-gram training with negative sampling

Let's glance at how we use it to train a basic model that predicts if two words appear together in the same context.

Preliminary steps

We start with the first sample in our dataset

input word	target word
not	thou
not	shalt
not	make
not	a
make	shalt
make	not
make	a
make	machine
a	not
a	make
a	machine
a	in
machine	make
machine	a
machine	in
machine	the
in	a
in	machine
in	the
in	likeness

not → Untrained Model

Task:
Predict neighbouring word

1) Look up embeddings

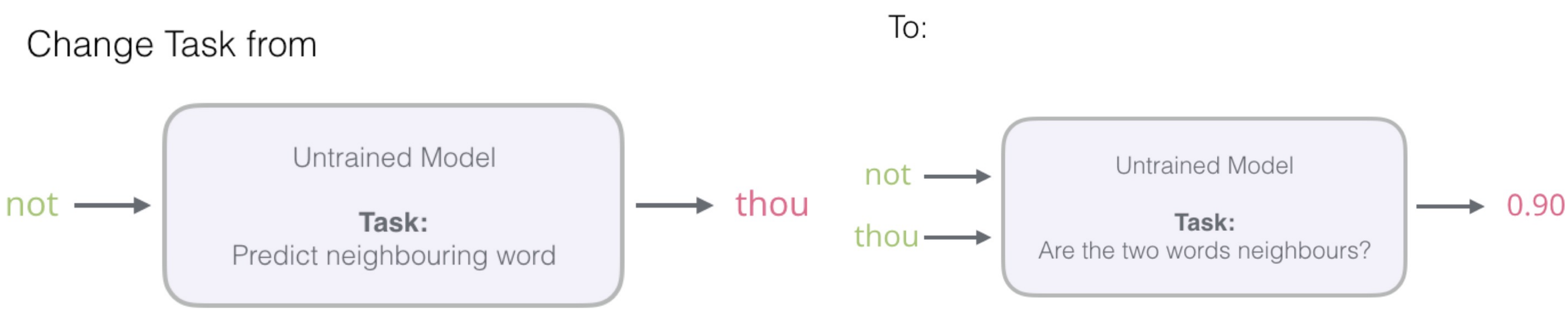
2) Calculate prediction

3) Project to output vocabulary

[Computationally Intensive]

Note on efficiency of negative sampling

We grab the feature and feed to the untrained model asking it to predict if the words are in the same context or not (1 or 0)



Negative examples

This can now be computed at blazing speed – processing millions of examples in minutes. But there's one loophole we need to close. If all of our examples are positive (target: 1), we open ourself to the possibility of a smartass model that always returns 1 – achieving 100% accuracy, but learning nothing and generating garbage embeddings.

input word	target word
not	thou
not	shalt
not	make
not	a
make	shalt
make	not
make	a
make	machine

input word	output word	target
not	thou	1
not	shalt	1
not	make	1
not	a	1
make	shalt	1
make	not	1
make	a	1
make	machine	1

Negative examples

For each sample in our dataset, we add **negative examples**. Those have the same input word, and a 0 label.

input word	output word	target
not	thou	1
not		0
not		0
not	shalt	1
not	make	1

Negative examples

We are contrasting the actual signal (positive examples of neighboring words) with noise (randomly selected words that are not neighbors). This leads to a great tradeoff of computational and statistical efficiency.

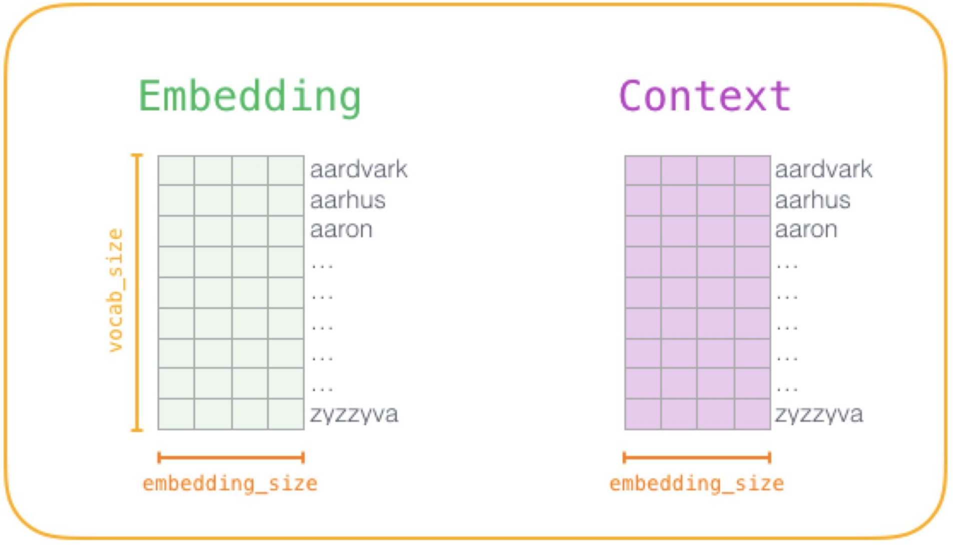
Training process

Now that we've established the two central ideas of skipgram and negative sampling, we can proceed to look closer at the actual word2vec training process.

- Before the training process starts, we **pre-process the text** we're training the model against. In this step, we determine the **size of our vocabulary** (we'll call this vocab_size, think of it as, say, 10,000) and which words belong to it.

- At the start of the training phase, we create **two matrices** – an Embedding matrix and a Context matrix. These two matrices have an **embedding for each word** in our vocabulary (So vocab_size is one of their dimensions). The second dimension is how long we want each embedding to be (**embedding_size** – 300 is a common value

Training process



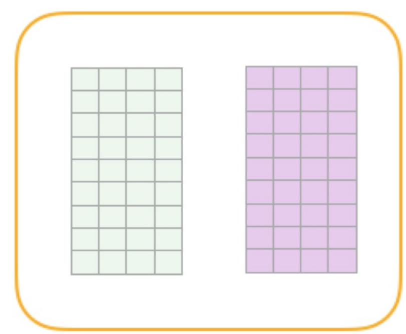
Training process

1. At the start of the training process, we **initialize** these matrices with **random values**. Then we start the training process. In each training step, **we take one positive example and its associated negative examples**. Let's take our first group:

dataset

input word	output word	target
not	thou	1
not	aaron	0
not	taco	0
not	shalt	1
not	mango	0
not	finglonger	0
not	make	1
not	plumbus	0
...	...	...

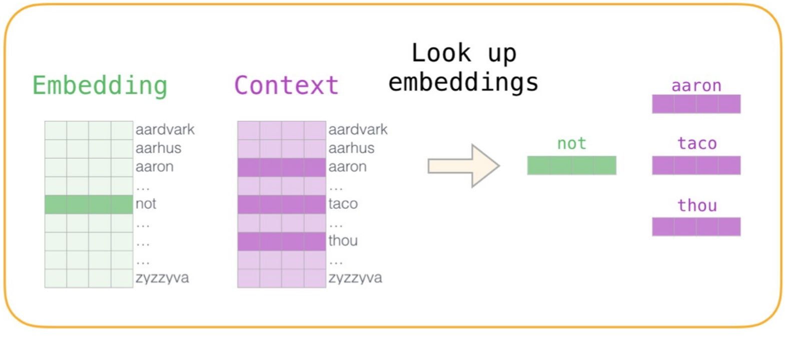
model



Training process

2. Now we have four words:
   - the input word *not*
   - the output/context words:
     thou (the actual neighbor), aaron, and taco (the negative examples).

We proceed to **look up their embeddings** – for the input word, we look in the Embedding matrix. For the context words, we look in the Context matrix (even though both matrices have an embedding for every word in our vocabulary)..



Training process

3. Then, we take the **dot product** of the input embedding with each of the context embeddings. In each case, that would result in a number, that number indicates the similarity of the input and context embeddings

4. Now we need a way to **turn these scores into something that looks like probabilities** – we need them to all be positive and have values between zero and one. This is a great task for sigmoid, the logistic operation. And we can now treat the output of the sigmoid operations as the model's output for these examples.

You can see that taco has the highest score and aaron still has the lowest score both before and after the sigmoid operations.

input word	output word	target	input • output	sigmoid()
not	thou	1	0.2	0.55
not	aaron	0	-1.11	0.25
not	taco	0	0.74	0.68

Training process

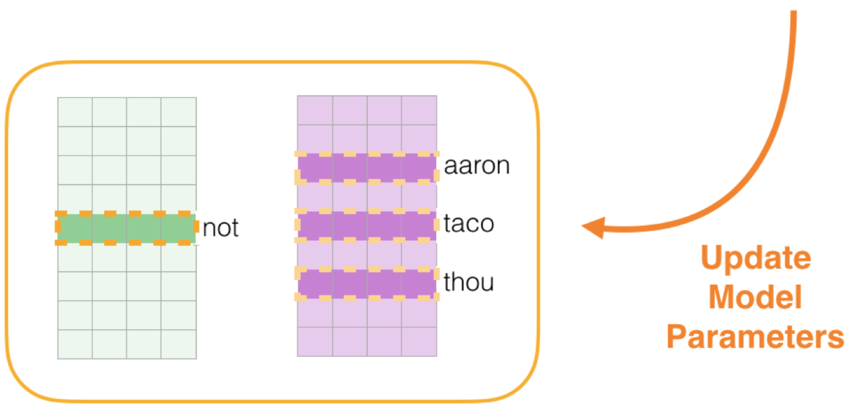
5. Now that the untrained model has made a prediction, and seeing as though we have an actual target label to compare against, let's calculate **how much error** is in the model's prediction. To do that, we just subtract the sigmoid scores from the target labels.

input word	output word	target	input • output	sigmoid()	Error
not	thou	1	0.2	0.55	0.45
not	aaron	0	-1.11	0.25	-0.25
not	taco	0	0.74	0.68	-0.68

Training process

6. Here comes the “learning” part of “machine learning”. We can now use this error score to **adjust the embeddings** of not, thou, aaron, and taco so that the next time we make this calculation, the result would be closer to the target scores

input word	output word	target	input • output	sigmoid()	Error
not	thou	1	0.2	0.55	0.45
not	aaron	0	-1.11	0.25	-0.25
not	taco	0	0.74	0.68	-0.68



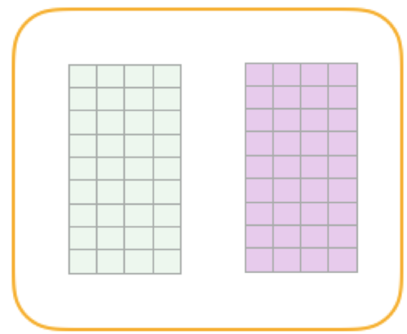
Training process

7. This concludes the training step. We emerge from it with slightly better embeddings for the words involved in this step (not, thou, aaron, and taco). We now proceed **to our next step** (the next positive sample and its associated negative samples) and do the same process again.

dataset

input word	output word	target
not	thou	1
not	aaron	0
not	taco	0
not	shalt	1
not	mango	0
not	finglonger	0
not	make	1
not	plumbus	0
...	...	...

model



The embeddings **continue to be improved while we cycle through our entire dataset** for a number of times. We can then stop the training process, discard the Context matrix, and use the Embeddings matrix as our pre-trained embeddings for the next task.

Outline

- Motivation
  - One-hot Encoding
  - Vectors and Documents
  - TF-IDF Vectors
  - PPMI Vectors
- Types of Word Vectors
  - Knowledge-based
  - Corpus-based
  - Word2Vec (CBOW)
  - Word2Vec (Skip-Gram)
  - **Others (FastText, Char-based,…)**
- Visualization and Evaluation

Other Language Units

- **Phrase**: Washington_Post is a newspaper
  Phrases can be automatically generated based on counts, e.g.,

$$\frac{count(w_i, w_j) - 6}{count(w_i) \rightarrow count(w_j)}$$

- **Character**: W a s h i n g t o n _ P o s t _ i s _ a _ n e w s p a p e r
  - Create a word representation from its character
  - Fully character level models

- **Sub-word:** Wash #ing #ton Post is a news #paper
  - N-grams, Byte Pair Encoding (BPE), Wordpiece, Sentencepiece

Based on context words II.2b

II.2 Direct prediction / Deep learning methods

fastText (Facebook, 2016):

subword-based skip-gram architecture: the vector representation of a word is the sum the embeddings of the character **n-grams** of the current word ($3 \le n \le 6$). Example: the fastTest representation of the word 'where' is the sum of 15 subwords (n-grams) embeddings:

3 grams: <wh, whe, her, ere, re>

4 grams: <whe, wher, here, ere>

5 grams: <wher, where, here>

6 grams: <where, where>

+ the word itself: <where>

Based on context words II.3

II.3 Hybrid: co-occurrence counts + prediction

GloVe: Global Vectors for Word Representation.

Ratios of word-word co-occurrence probabilities have the potential for encoding some form of 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

GloVe: Global Vectors for Word Representation
Jeffrey Pennington, Richard Socher, Christopher D. Manning

Based on context words II.3

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

The GloVe model is trained on the non-zero entries of a global word-word co-occurrence matrix, which tabulates how frequently words co-occur with one another in a given corpus.

The training objective is to learn word vectors such that their dot product equals the logarithm of the words' probability of co-occurrence. (ratio equals difference of logs)

GloVe: Global Vectors for Word Representation
Jeffrey Pennington, Richard Socher, Christopher D. Manning

Outline

- Motivation
  - One-hot Encoding
  - Vectors and Documents
  - TF-IDF Vectors
  - PPMI Vectors
- Types of Word Vectors
  - Knowledge-based
  - Corpus-based
  - Word2Vec (CBOW)
  - Word2Vec (Skip-Gram)
  - Others (FastText, Char-based,…)
- **Visualization and Evaluation**

Example

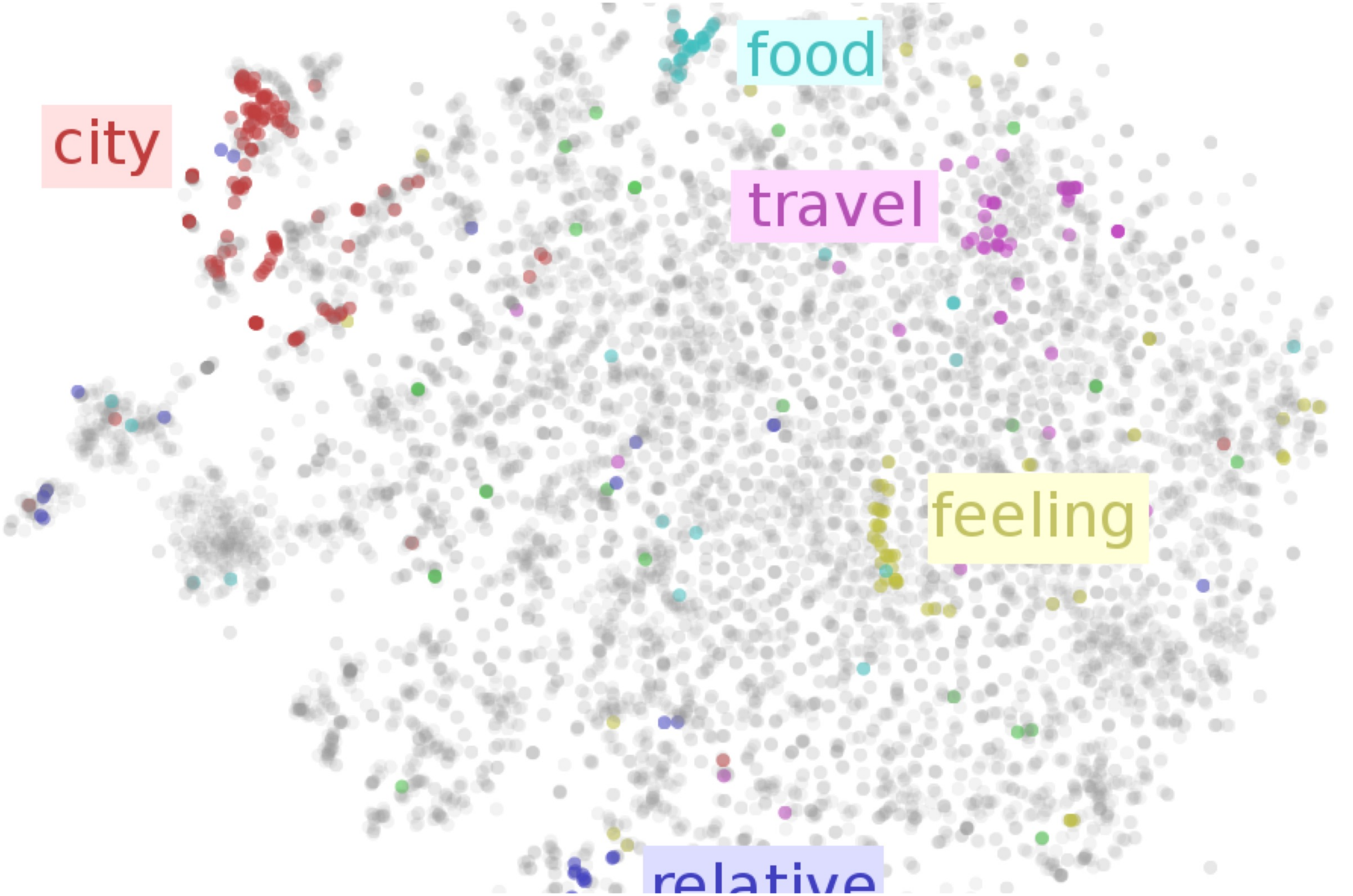
Closest words to the target word frog

frog	(rana, granota)
frogs	(ranas, granotes)
toad	(sapo, gripau)
litoria	(litoria, litòria)
leptodactylidae	
rana	
lizard	(lagartija, sargantana)
eleutherodactylus	



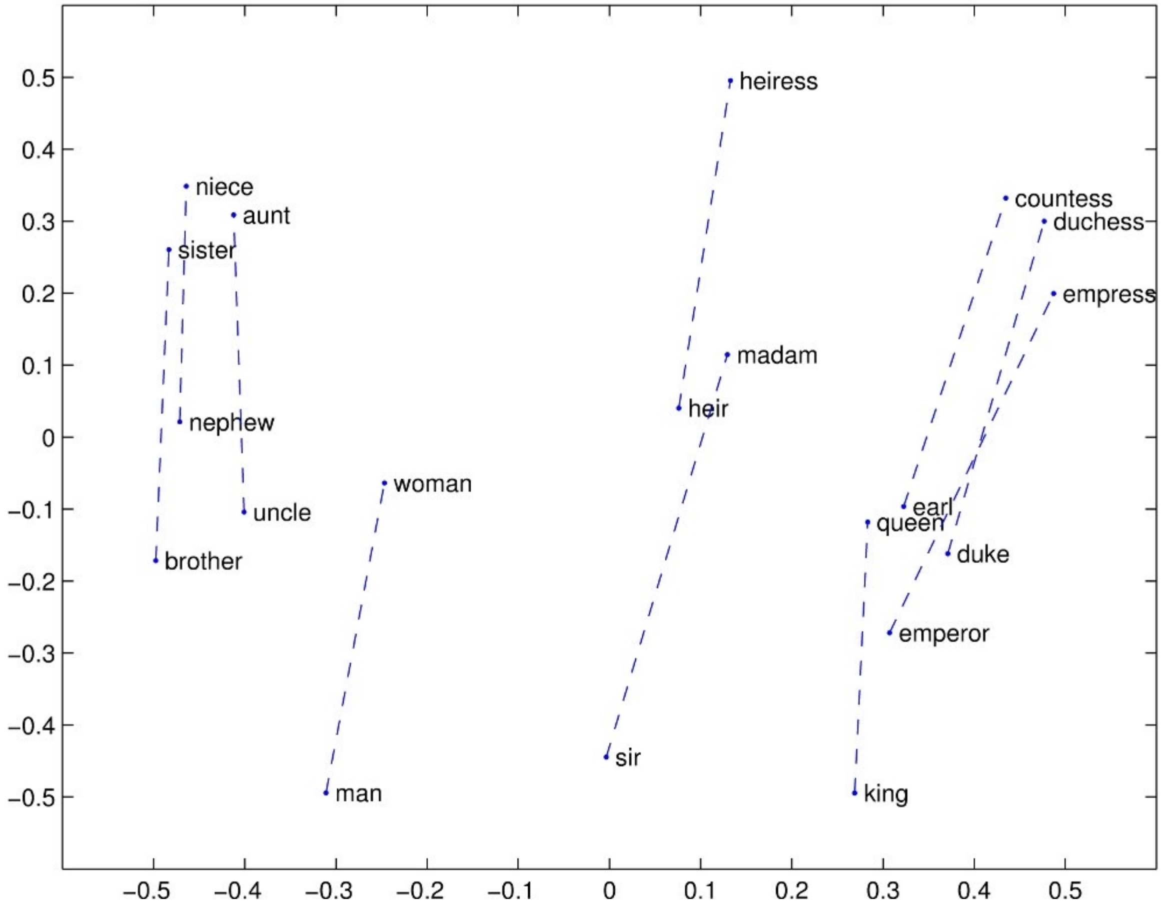

GloVe: Global Vectors for Word Representation
Jeffrey Pennington, Richard Socher, Christopher D. Manning

Visualizing Representations

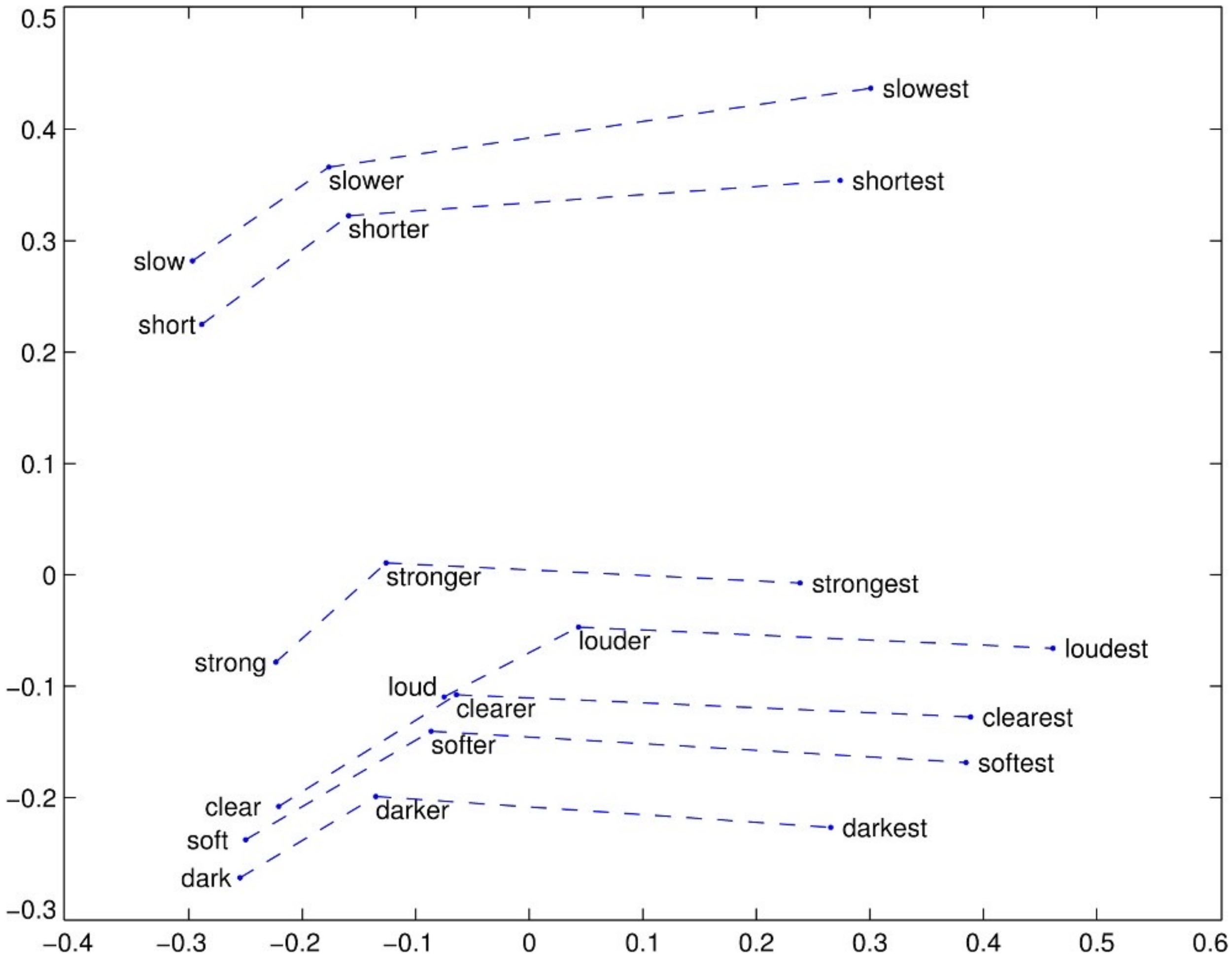


Christopher Olah

Example: Linear structures man-woman



Example: Linear structures comparative - superlative



Example: Catalan word vectors (CBOW)

‘dimecres’ + (‘dimarts’ – ‘dilluns’) = ‘dijous’

‘tres’ + (‘dos’ – ‘un’) = ‘quatre’

‘tres’ + (‘2’ – ‘dos’) = ‘3’

‘viu’ + (‘coneixia’ – ‘coneix’) = ‘vivia’

‘la’ + (‘els’ – ‘el’) = ‘les’

‘Polònia’ + (‘francès’ – ‘França’) = ‘polonès’

Question

- How can we evaluate word vectors?

Evaluation

- Intrinsic vs Extrinsic evaluation
  - Properly evaluating the Word vectors (similarity, analogy, distance)
  - Vs. Downstream tasks (translation, sentiment analysis)...

Intrinsic Evaluation

Word similarity:

Closest word to w_c

$$\cos(w_x, w_y) = \frac{w_x \cdot w_y}{||w_x|| \; ||w_y||}$$

Word analogy:

a is to *b* as *c* is to ….

Find *d* such as w_d is closest to $w_c + (w_b - w_a)$

- Athens is to Greece as Berlin to ….
- Dance is to dancing as fly to ….

"Distance":

Cosine similarity (normalized dot product)
Euclidean distance
Dot product

Challenges of Word Vectors

- Mention a few

Summary

- Meaning Word Embedding

"Any technique mapping a word (or phrase) from it's original high-dimensional input space (the body of all words) to a lower-dimensional numerical vector space - so one embeds the word in a different space"

- Importance of Word Embedding

"Word representations are a critical component of many natural language processing systems."

Take home message

- Similarity in meaning similarity in vectors Mathematics should be able to encode meaning
- You shall know a word by the company it keeps ;) The environment of a word gives meaning to it
- Use BIG datasets (millions of billions to words) Especially neural models require lots of data!