WORD VECTORS

Slides are adapted from Marta R. Costa-jussà, José A. R. Fonollosa from UPC

Outline

- Motivation
- Types of Word Vectors
- Visualization and Evaluation

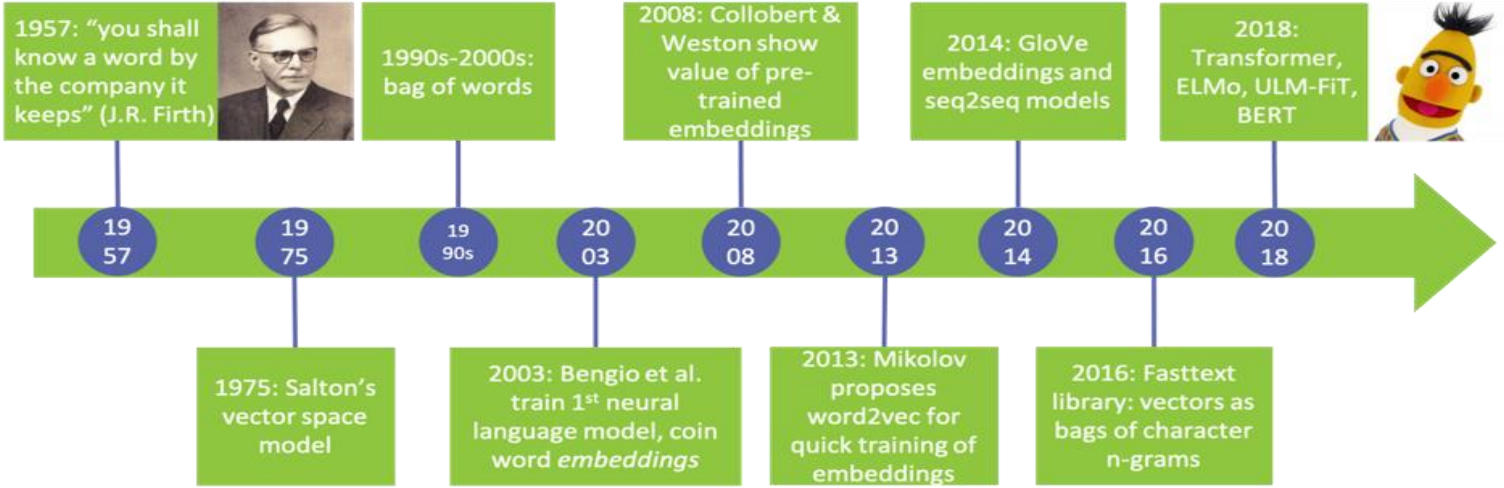
Question

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

A Word embedding is a numerical representation of a word

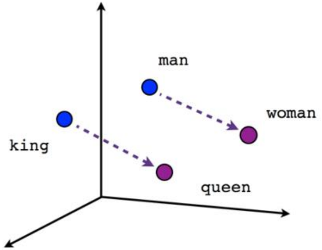
- Word embeddings allow for arithmetic operations on a text
  - Example: time + flies
- Word embeddings have been referred also as:
  - Semantic Representation of Words
  - Word Vector Representation

Timeline

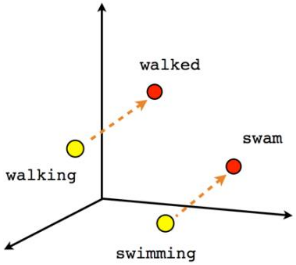


1957: "you shall know a word by the company it keeps" (J.R. Firth)
1990s-2000s: bag of words
2008: Collobert & Weston show value of pre-trained embeddings
2014: GloVe embeddings and seq2seq models
2018: Transformer, ELMo, ULM-FiT, BERT
19 57
19 75
19 90s
20 03
20 08
20 13
20 14
20 16
20 18
1975: Salton's 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

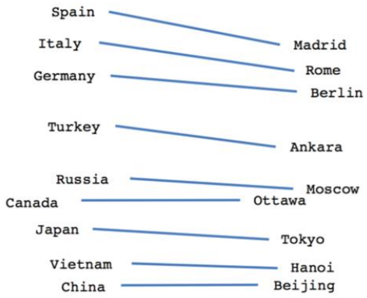
Word vectors



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)

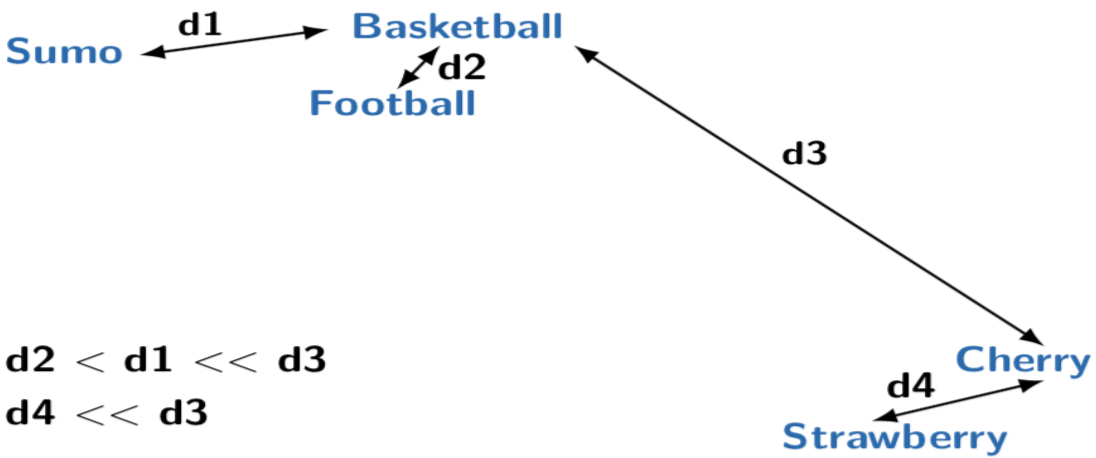
- Words that occur in similar contexts tend to have similar meaning (Harris, 1954)

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
- Types of Word Representation
- Visualization and Evaluation

How to represent a word

Machines do not understand text the way we do. To enable NLP models to work with text, we need to represent it in a numerical format.

Common Text Representation Techniques:

1. **Bag of Words (BoW)**
    1. Text is represented as a set of words, ignoring grammar and word order.
    2. Each unique word in a corpus forms a feature, with its frequency as the value.
2. **TF-IDF (Term Frequency-Inverse Document Frequency)**
    1. Enhances BoW by considering the importance of words across documents.
    2. Gives higher weight to rare but significant words, reducing the weight of common words

How to represent a word

3. **Word Embeddings (e.g., Word2Vec, GloVe)**
   3. Dense vector representations that capture semantic meaning.
   4. Words with similar meanings have similar vector representations.
4. **Contextual Embeddings (e.g., BERT, GPT)**
   1. Advanced embeddings that capture word meaning based on the context of the sentence.
   2. Dynamic word representations that change with different contexts.

Word vectors (Word Embeddings)

- Use the many contexts of *w* to build up a representation of *w*

…government debt problems turning into ***banking*** *crises as happened in 2009…*

…saying that Europe needs unified ***banking*** *regulation to replace the hodgepodge…*

…India has just given its ***banking*** *system a shot in the arm…*

These context words will represent ***banking***

We will build a dense vector for each word, chosen so that it is similar to vectors of words that appear in similar contexts.

$$\textit{banking} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

■ **Based on context words (**Word2vec)

Word2vec (Mikolov, Google 2013) is a framework for learning word vectors

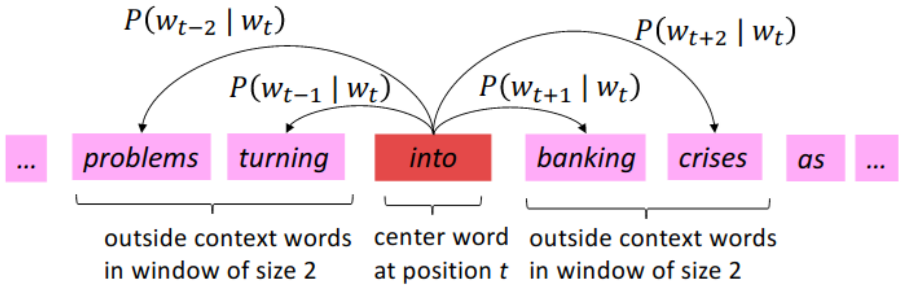
- We have a large corpus ("body") of text
- Every word in a fixed vocabulary is represented by a vector
- Go through each position t in the text, which has a center word c and context ("outside") words o
- Use the similarity of the word vectors for c and o to calculate the probability of o given c (or vice versa)
- Keep adjusting the word vectors to maximize this probability

Based on context words II.2a

Word2vec (Mikolov, Google 2013) two models:

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

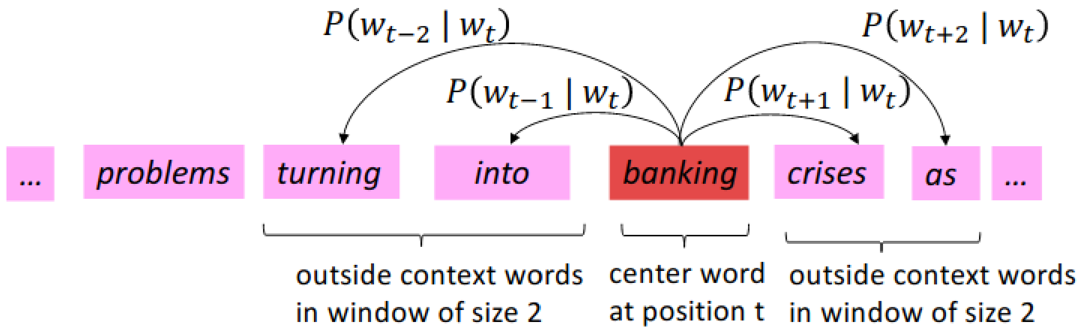
Example windows and process for computing $P(w_{t+j} \mid w_t)$



Skipgram illustration is taken from slides from stanford slides

Based on context words II.2a

Example windows and process for computing $P(w_{t+j} \mid w_t)$



Word2vec: objective function

For each position $t = 1, \dots, T$, predict context words within a window of fixed size m, given center word w_j. Data likelihood:

$$\text{Likelihood} = L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \ne 0}} P(w_{t+j} \mid w_t; \theta)$$

θ is all variables to be optimized

sometimes called a *cost* or *loss* function

The objective function $J(\theta)$ is the (average) negative log likelihood:

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \ne 0}} \log P(w_{t+j} \mid w_t; \theta)$$

Minimizing objective function $\Leftrightarrow$ Maximizing predictive accuracy

Word2vec: objective function

- We want to minimize the objective function:

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\le j\le m\\ j\neq 0}} \log P\left(w_{t+j} \mid w_t; \theta\right)$$

- **Question:** How to calculate $P\left(w_{t+j} \mid w_t; \theta\right)$?

- **Answer:** We will *use two* vectors per word *w*:
  - v_w when *w* is a center word
  - u_w when *w* is a context word

- Then for a center word *c* and a context word *o*:

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w\in V}\exp(u_w^T v_c)}$$

Word2vec: Prediction function

SOFTMAX

② Exponentiation makes anything positive

$$P(o|c) = \frac{\exp(\boxed{u_o^T v_c})}{\boxed{\sum_{w \in V} \exp(u_w^T v_c)}}$$

① Dot product compares similarity of o and c.

$u^T v = u.v = \sum_{i=1}^{n} u_i v_i$

Larger dot product = larger probability

③ Normalize over entire vocabulary
to give probability distribution

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

Keep in mind: it is simplified, using sigmoid (in reality softmax should be better used)

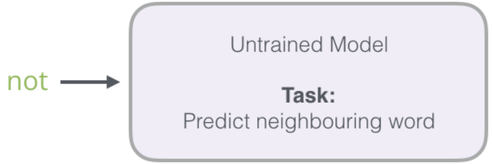
We will add a small new concept (negative sampling)

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



1) Look up embeddings

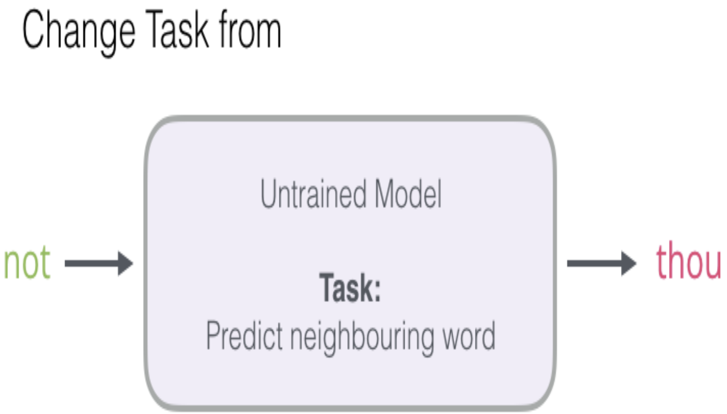
2) Calculate prediction

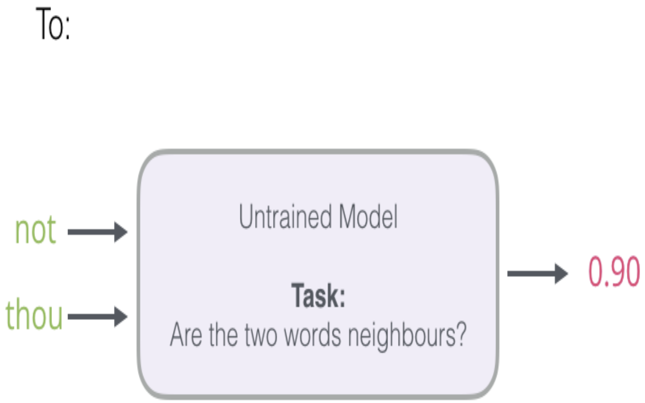
3) Project to output vocabulary

[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.

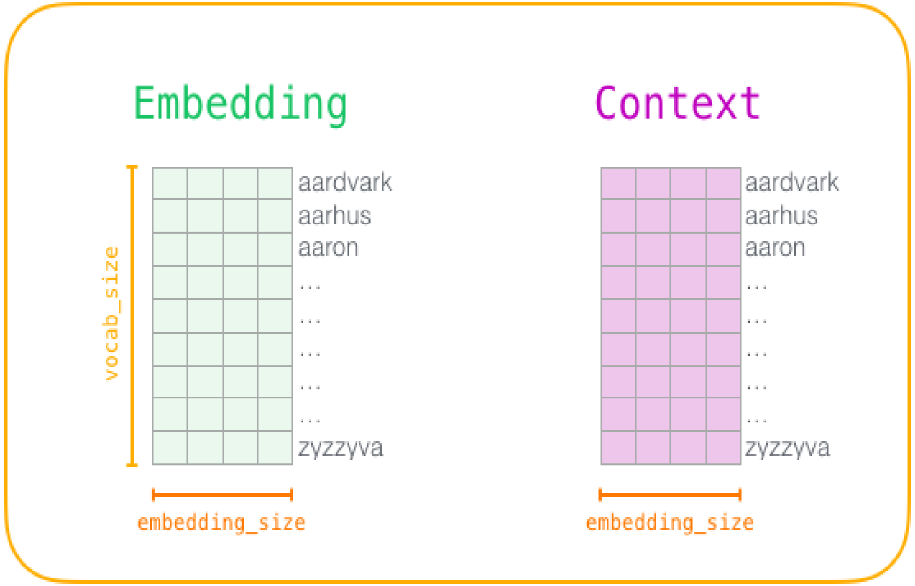
Training process

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



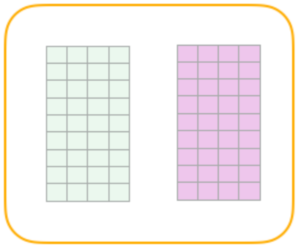
Training process

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

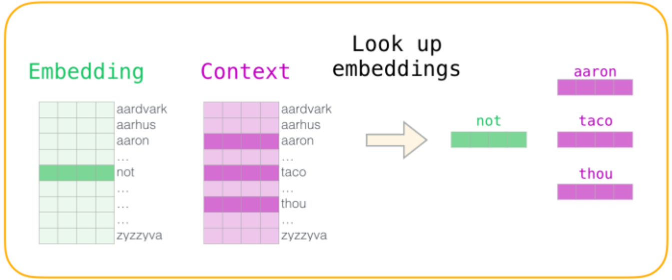


Training process

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

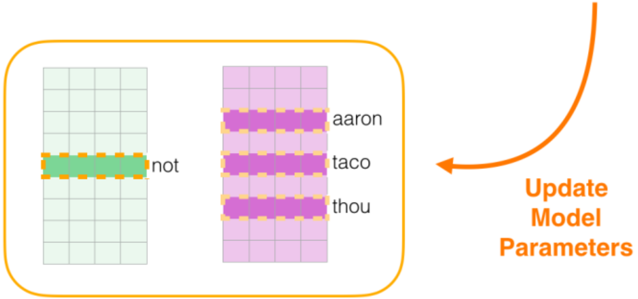
5. Now that the untrained model has made a prediction, and seeing as though we have an actual target label to compare against, let's calculate **how much error** is in the model's prediction. To do that, we just subtract the sigmoid scores from the target labels.

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



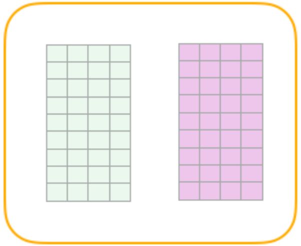
Training process

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.

Based on context words II.2b

Word2vec (Mikolov, Google 2013) second model:

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

CBOW

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

Window of 5 words

left window of size 2

right window of size 2

Continuous bag-of-words (CBOW)

FUN WITH FILL-INS

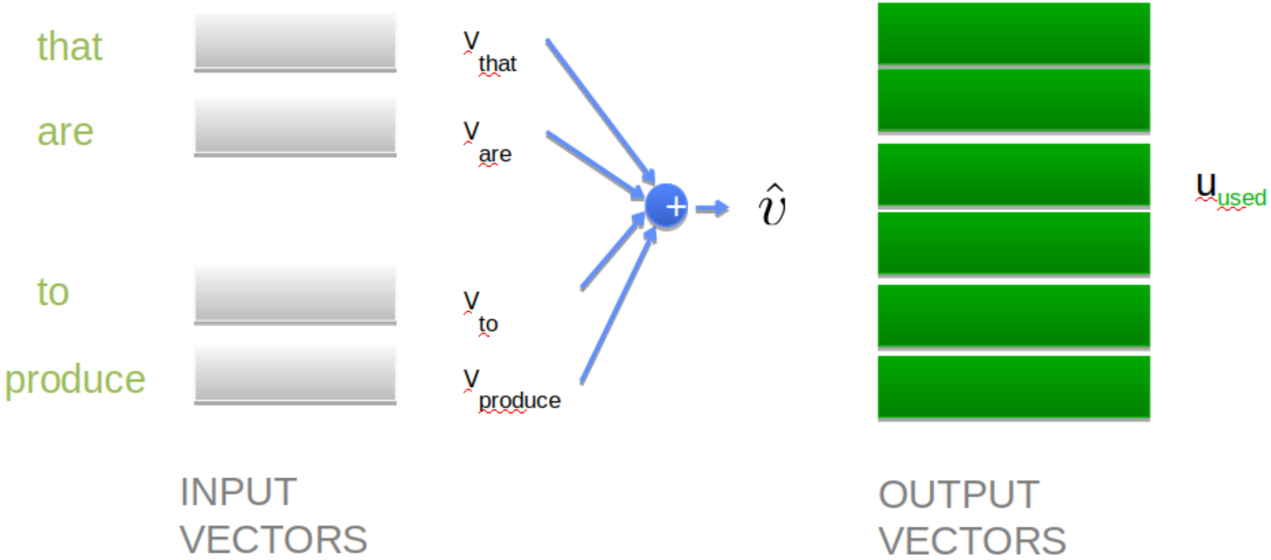
First Grade Sight Words

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.2b



CBOW

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

Other Language Units

- **Phrase**: Washington_Post is a newspaper
  Phrases can be automatically generated based on counts, e.g.,
  $$\frac{count(w_i, w_j) - \delta}{count(w_i) \times count(w_j)}$$
- **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 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 ($3 \leq n \leq 6$). Example: the fastText 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

Word embedding properties

- Similar words tend to have similar embeddings or vectors.
- Since words are represented as real valued dense vectors, the similarity between them can be measured using the cosine similarity measure.

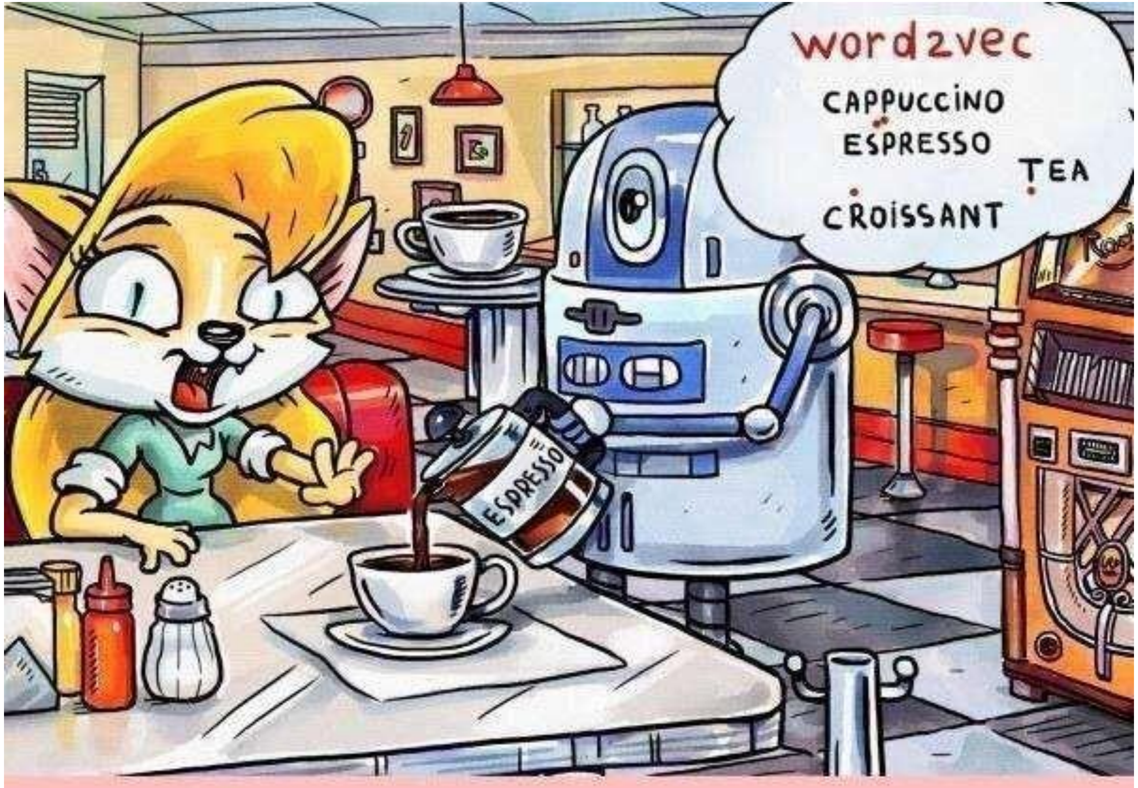
```
model.wv[u'مصر']
array([ 4.96487588e-01,  1.81284651e-01,  1.01665771e+00,
        3.05797219e+00,  2.69687176e-01, -4.95693743e-01,
       -3.91436696e+00,  1.19369626e-01,  1.05419767e+00,
        1.37945485e+00, -1.66222382e+00,  5.26237428e-01,
       -5.06798863e-01, -8.94690096e-01, -1.14547145e+00,
       -1.17323422e+00, -1.91685811e-01, -6.46347478e-02,
        8.14857781e-01,  5.63521564e-01,  1.01283193e+00,
       -1.65220642e+00, -2.95289844e-01, -5.86339235e-01,
       -7.70911515e-01, -8.79850328e-01, -1.83659840e+00,
       -1.17567265e+00, -5.19872069e-01, -1.77343929e+00,
        1.36198223e+00,  2.74382854e+00, -1.32640815e+00,
        1.38219535e+00, -1.17181027e+00, -9.99460280e-01,
        5.28308094e-01,  9.80325341e-01,  1.80007434e+00,
       -4.78034377e-01, -5.39939106e-01, -1.49182498e+00,
       -1.45962775e+00,  6.53993845e-01,  7.85182953e-01,
       -8.04159164e-01,  4.30263996e-01,  1.40719235e-01,
        2.50109267e+00,  1.47388113e+00,  1.60538805e+00,
       -1.00153041e+00,  1.61531591e+00,  2.45009112e+00,
```

The previous example revisited

If the documents

1. I love cake
2. I love cupcakes
3. I love Cairo

are now represented as the sum of their word vectors, the cosine similarity between document 1 and document 2 will be: 0.78 and the cosine similarity between document 1 and document 3 will be: 0.48



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

Examples of semantic expressiveness

The famous example of King – Man + Woman = Queen Relationship pairs:

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

Relationship pairs in a word embedding. From Mikolov *et al.* (2013b).

Outline

- Motivation
- Types of Word Vectors
- Visualization and Evaluation

Example

Closest words to the target word frog

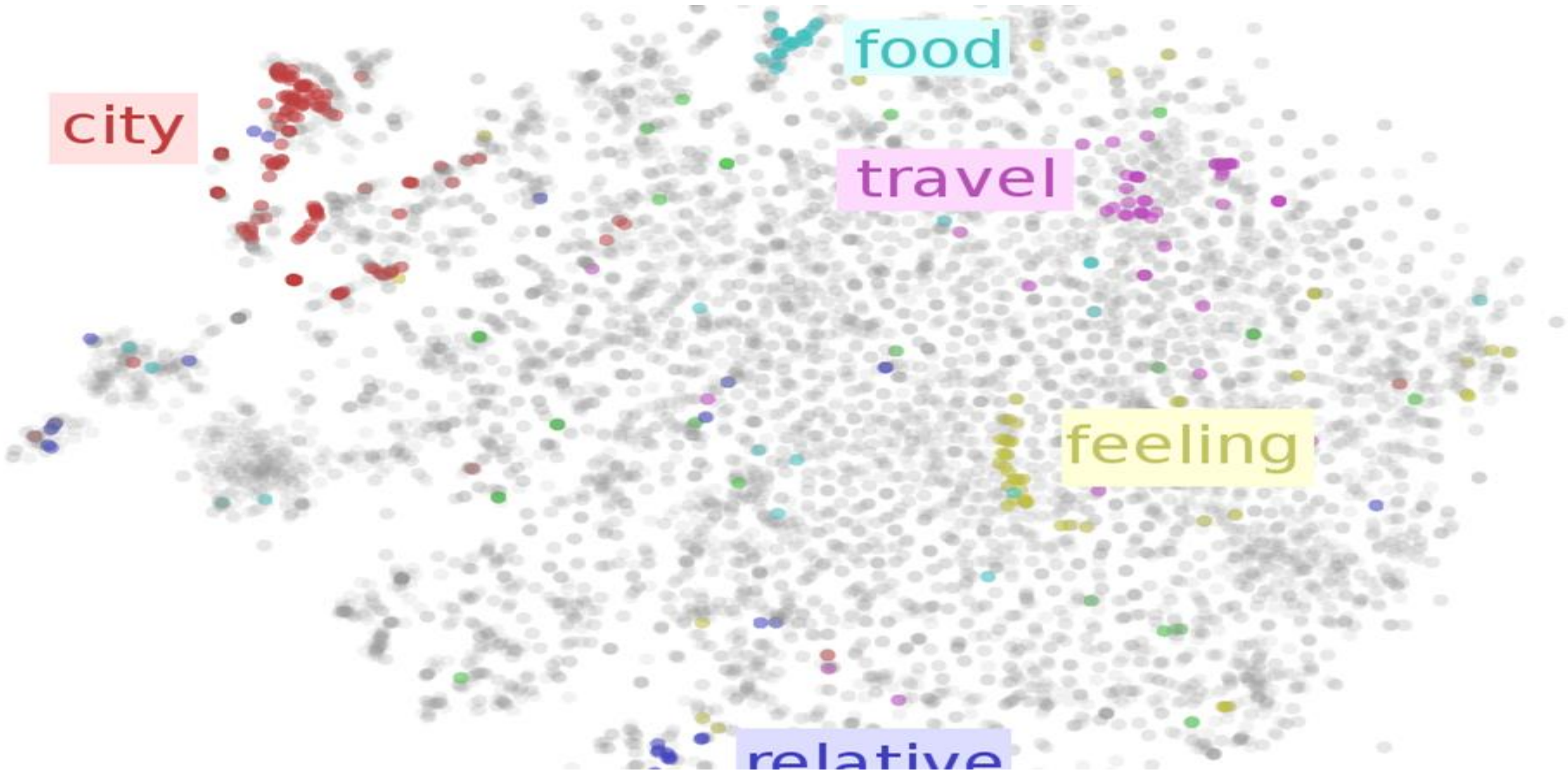
frog	(rana, granota)
frogs	(ranas, granotes)
toad	(sapo, gripau)
litoria	(litoria, litòria)
leptodactylidae	
rana	
lizard	(lagartija, sargantana)
eleutherodactylus	




GloVe: Global Vectors for Word Representation

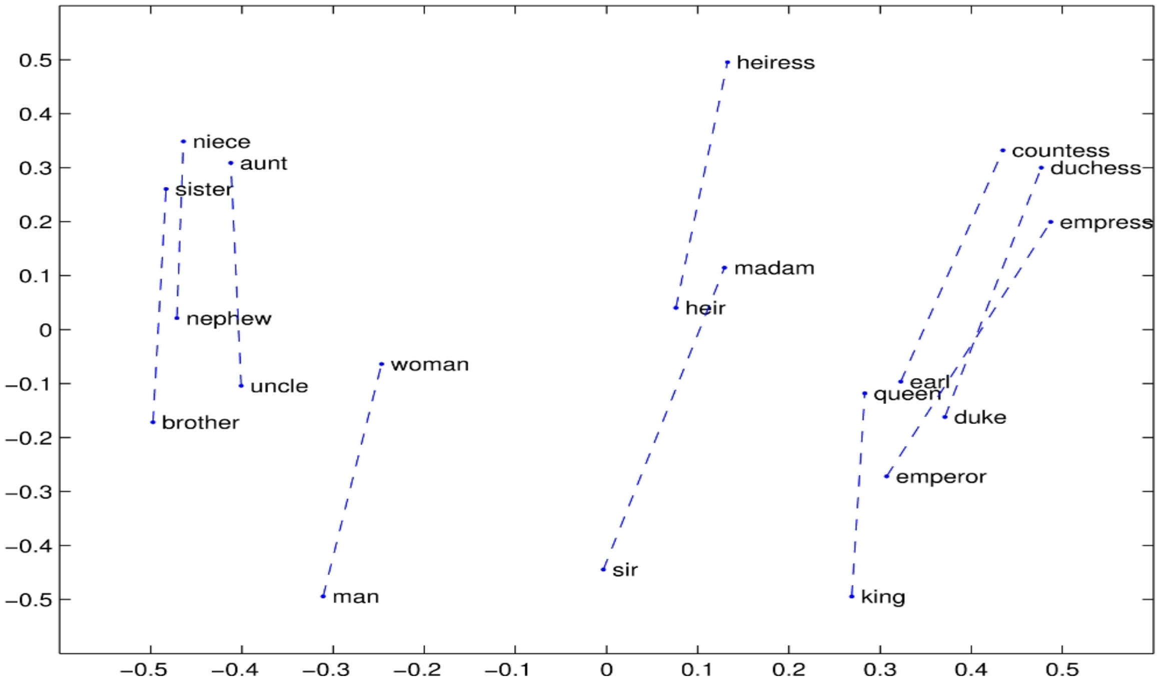
Jeffrey Pennington, Richard Socher, Christopher D. Manning

Visualizing Representations

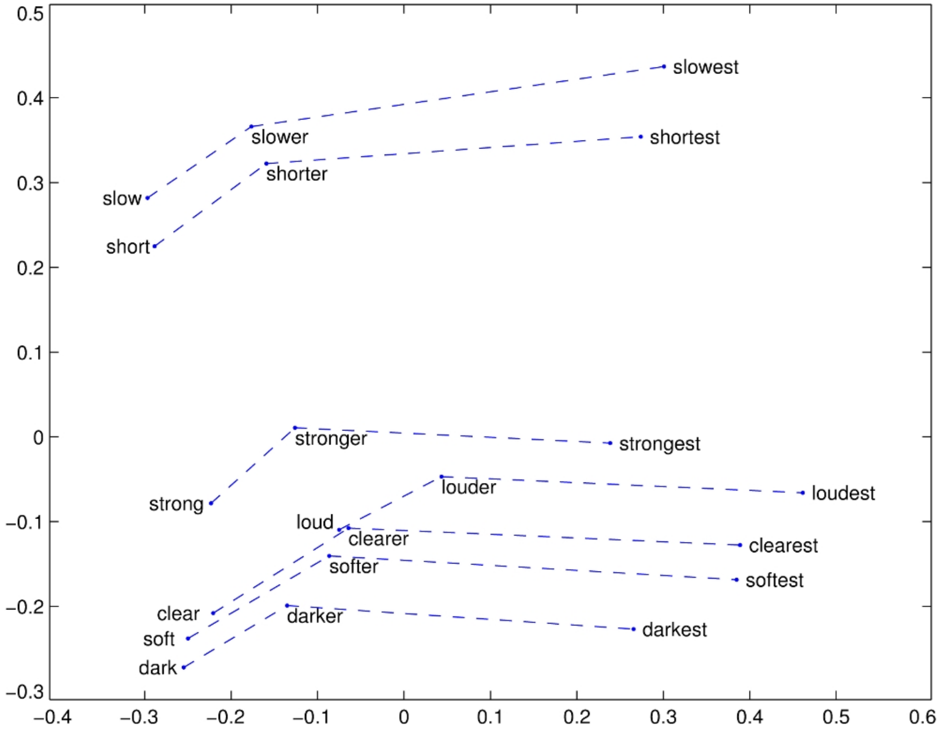


Christopher Olah

Example: Linear structures man-woman



Example: Linear structures comparative - superlative



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!

Tools

- Gensim (https://radimrehurek.com/gensim/): A python based tool that offers Word2Vec, LSI, and LDA implementations.
- GloVe (https://nlp.stanford.edu/projects/glove/) – Standford's tool for generation word embedding
- FastText - "a library for efficient learning of word representations and sentence classification".
- Word2Vec (https://code.google.com/archive/p/word2vec/): The original Google implementation of the Word2Vec algorithm. Code is written in C.
- Word2Vec in Java (https://deeplearning4j.org/word2vec)

Pre-trained Embeddings

- Embeddings for 294 languages, trained on Wikipedia using fastText https://github.com/facebookresearch/fastText/blob/master/pretrained-vectors.md
- Pre-trained vectors trained on part of Google News dataset (about 100 billion words) using word2Vec. The model contains 300-dimensional vectors for 3 million words and phrases. https://goo.gl/RhkUE8
- GloVe vectors trained on Wikipedia 2014,and Gigaword 5 (6B tokens, 400K vocab, uncased, 50d, 100d, 200d, & 300d vectors), http://nlp.stanford.edu/data/glove.6B.zip
- GloVe vectors trained on Twitter (2B tweets, 27B tokens, 1.2M vocab, 25d, 50d, 100d, & 200d vectors): http://nlp.stanford.edu/data/glove.twitter.27B.zip
- AraVec: vectors trained on Twitter and Wikipedia using approximately 1,169,075,128 tokens. https://github.com/bakrianoo/aravec/

More stuff for you (optional) ;)

I want more references:

- https://web.stanford.edu/~jurafsky/slp3/6.pdf
- https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1214/readings/cs224n-2019-notes01-wordvecs1.pdf
- https://lena-voita.github.io/nlp_course/word_embeddings.html

I want to play ;)

https://projector.tensorflow.org/