Department of Linguistics and Translation
香港城市大學
City University of Hong Kong

Computational Linguistics
LT3233

Jixing Li

Lecture 10: Word Embeddings

Slides adapted from Dan Jurafsky

Lecture plan

- Representing word meaning
- tf-idf
- Word2Vec: skip-gram
- Evaluating word embeddings
- Short break (15 mins)
- Hands-on exercises

Final exam

- 6:30-8:30 pm, Dec 9, LI-G600

Representing name as features

chinese_name	major	gender	n1_male	n2_male	n1_uniqueness	n2_uniqueness
林加敏	LLA	F	0.442	-0.562	2.795	2.087

x = [0.442,-0.562,2.795,2.087]

W1 = [[1,3,2,4],
[2,1,4,3]]

W2 = [-1,2]

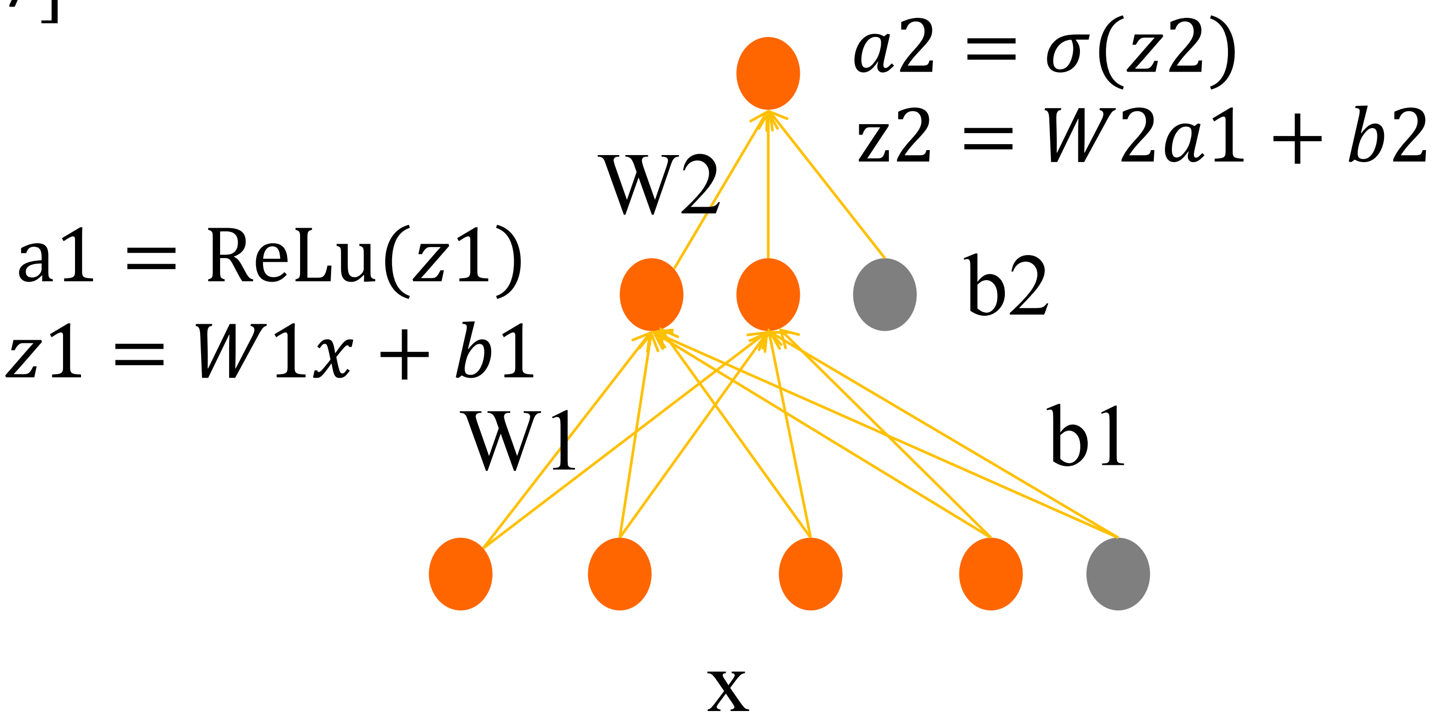
b1 = [1,-1], b2 = 2

z1 = W1x+b1

a1 = ReLU(z1) = max(z1,0)

z2 = W2a1+b2

a2 = σ(z2)=1/(1+e^{-z2})



Word meaning: attributes

Binder et al. (2016): 65 dimensions, scale: 0-6

Word	Vision	Bright	Dark	Color	Pattern	Large	Small
ant	3.5484	0.3548	3.5806	3.9355	1.9355	0.0968	5.871
bicycle	5.3	1.1667	0.6333	1	2.1667	1.7	1.2667
farm	5.7097	1.1935	0.5161	1.7419	1.8065	5.0645	0.129
farmer	4.1786	0.5	0.3214	0.4286	0.6071	1.4286	0.6786
green	4.2963	1.7778	1	5.9259	1.5926	0.1852	0.1111
red	5	3.2857	1.25	6	1.4643	0.1071	0.0357
rocket	5.5	2.9333	0.7333	1.8667	1.9	5.6	0.2333
trust	0.3793	0.1379	0.0345	0.3103	0.2069	0.3103	0.069

Word meaning: co-occurrence

Wittgenstein (1953): The meaning of a word is its use in the language

Harris (1954): If A and B have almost identical environments we say that they are synonyms.

Firth (1957): A word is characterized by the company it keeps.

Example: *ongchoi*

Suppose you see these sentences:

- ongchoi is delicious sautéed with garlic.
- ongchoi is superb over rice
- ongchoi leaves with salty sauces

And you've also seen these:

- …spinach sautéed with garlic over rice
- chard stems and leaves are delicious
- collard greens and other salty leafy greens

Conclusion:

- ongchoi is a leafy green like spinach, chard, or collard greens

We could conclude this based on words like "leaves" and "delicious" and "sauteed"

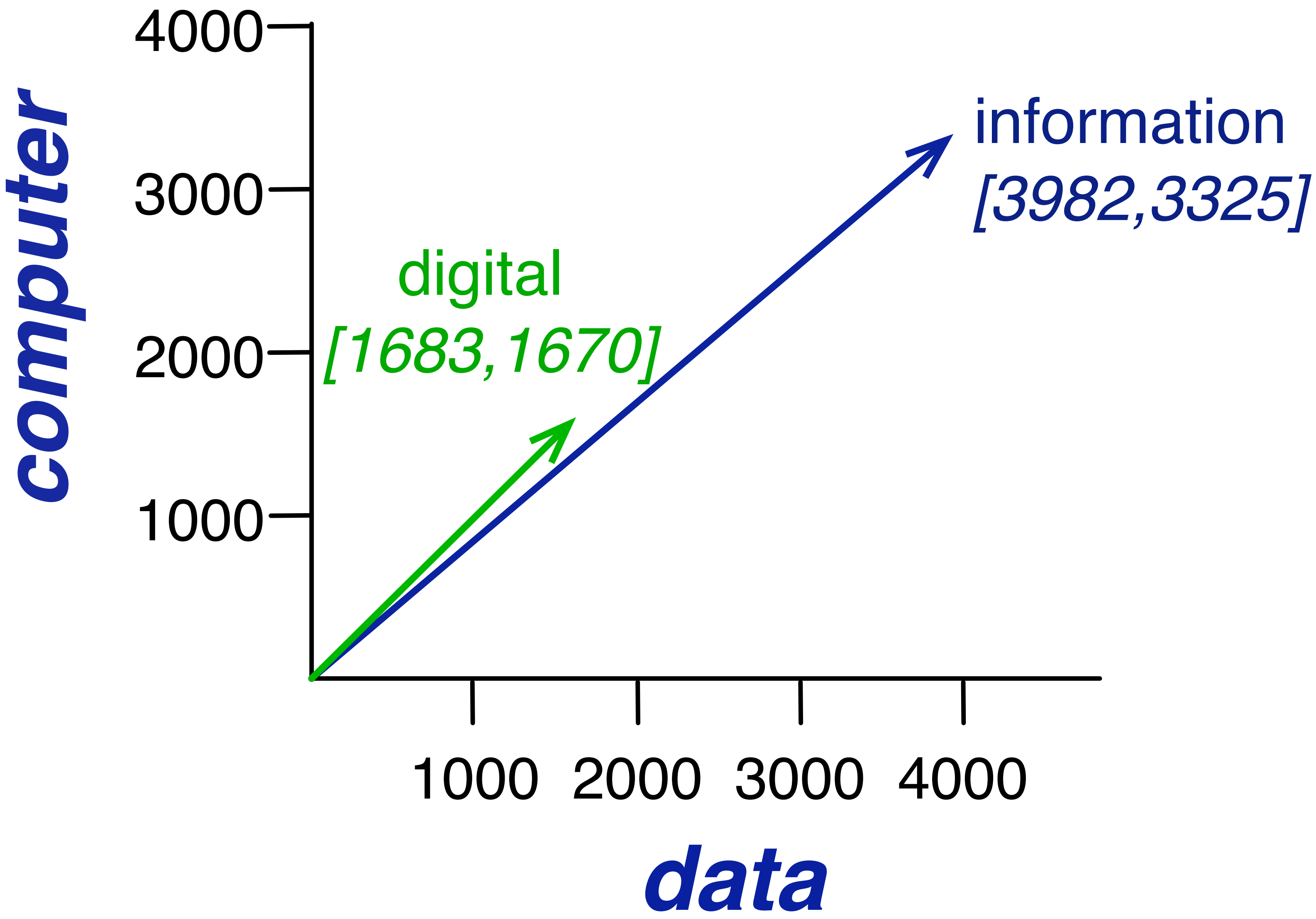
Defining meaning by linguistic distribution

Two words are similar in meaning if their contexts are similar

is traditionally followed by	**cherry**	pie, a traditional dessert
often mixed, such as	**strawberry**	rhubarb pie. Apple pie
computer peripherals and personal	**digital**	assistants. These devices usually
a computer. This includes	**information**	available on the internet

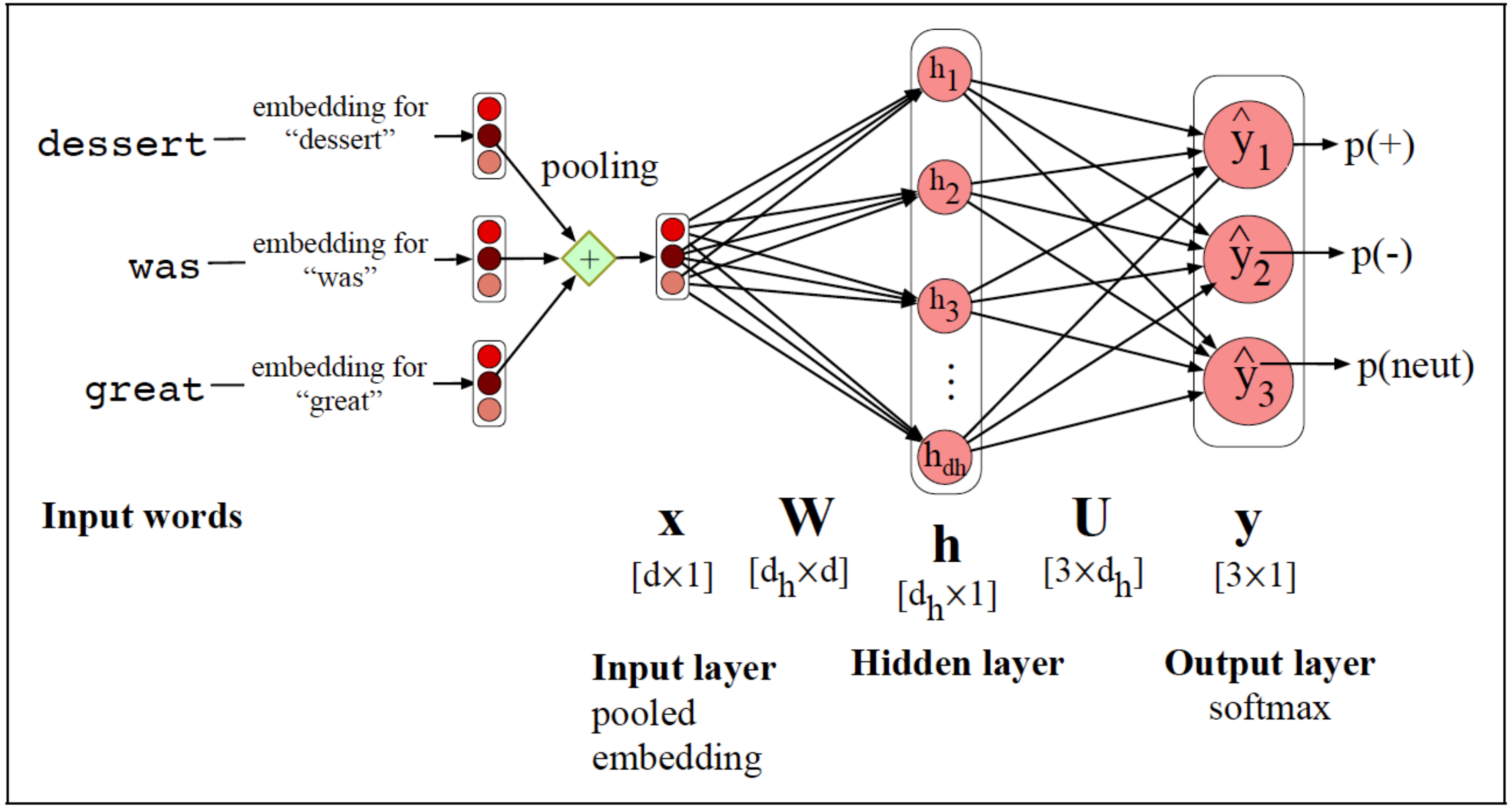
	aardvark	...	**computer**	**data**	**result**	**pie**	**sugar**	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...

Word as vector in space



Word embeddings

A word vector is called an "embedding" because it's embedded into a space. → Every modern NLP algorithm uses embeddings as the representation of word meaning



Why embeddings?

Can generalize to **similar but unseen** words!

ongchoi and spinach will have similar embeddings

tf-idf

term frequency-inverse document frequency: Words are represented by (a simple function of) the **counts** of nearby words.

term-context matrix: context window = 4

	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...
good	1085	...	4300	5638	5283	4828	3968	...

Term frequency (tf)

$\mathbf{tf}_{t,d}$ = count(*t*,*d*): the frequency of word ***t*** in document ***d***

Instead of using raw count, we squash a bit using ***log10***

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

	aardvark	**…**	**computer**	**data**	**result**	**pie**	**sugar**	**…**
cherry	0	…	0.48	0.95	1	2.65	1.41	…
strawberry	0	…	0	0	0.3	1.79	1.30	…
digital	0	…	3.22	3.23	1.93	0.78	0.70	…
information	0	…	3.52	3.60	2.58	0.78	1.15	…
good	3.04	…	3.63	3.75	3.72	3.68	3.60	…

Inverse document frequency (idf)

df_t: the number of documents ***t*** occurs in.

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

N is the total number of documents in the collection

	df	**idf**
cherry	2800	2.55
strawberry	3005	2.52
digital	7603	2.12
information	14378	1.84
good	275423	0.56

N=1000000

good co-occurs with many words, so its idf will be small

Final tf-idf weighted value for a word

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

	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	1.22	2.42	2.55	6.76	3.60	...
strawberry	0	...	0	0	0.76	4.51	3.28	...
digital	0	...	6.83	6.85	4.09	1.65	1.48	...
information	0	...	6.48	6.62	4.75	1.44	2.12	...
good	1.70	...	2.03	2.1	2.08	2.06	2.02	...

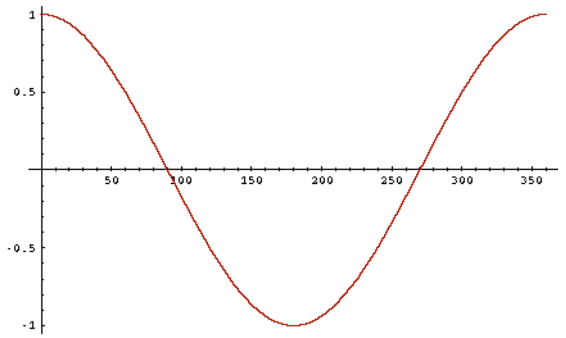
Computing word similarity: Cosine

$$\text{cosine}(\vec{v},\vec{w}) = \frac{\vec{v}\cdot\vec{w}}{|\vec{v}||\vec{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

$$= v_1 w_1 + v_2 w_2 + \ldots + v_N w_N$$

Normalized by the length of the vector

The dot product tends to be high when the two vectors have large values in the same dimensions
→ a useful similarity metric between vectors



-1: vectors point in opposite directions: dissimilar
+1: vectors point in same directions: similar
0: vectors are orthogonal

Cosine similarity: Example

$$\cos\left(\frac{\vec{v}\cdot\vec{w}}{|\vec{v}||\vec{w}|}\right) = \frac{\vec{v}}{|\vec{v}|}\cdot\frac{\vec{w}}{|\vec{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

	pie	data	computer
cherry	6.76	2.42	1.22
digital	1.65	6.85	6.83
information	1.44	6.62	6.48

$$\cos(cherry, information) = \frac{6.76 * 1.44 + 2.42 * 6.62 + 1.22 * 6.48}{\sqrt{6.76^2 + 2.42^2 + 1.22^2}\sqrt{1.44^2 + 6.62^2 + 6.48^2}} = 0.49$$

$$\cos(digital, information) = \frac{1.65 * 1.44 + 6.85 * 6.62 + 6.83 * 6.48}{\sqrt{1.65^2 + 6.85^2 + 6.83^2}\sqrt{1.44^2 + 6.62^2 + 6.48^2}} = 0.99$$

semantically-related words have higher cosine similarity

Sparse vs dense vectors

tf-idf vectors are

- **long** (length |V|= 100000)
- **sparse** (most elements are zero)

Alternative: learn vectors which are

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

→ Short vectors may be easier to use as features in machine learning (fewer weights to tune)

Word2Vec (Mikolov et al., 2013): simple static embeddings
https://code.google.com/archive/p/word2vec/

Word2Vec

- Popular embedding method
- Very fast to train
- Code available on the web

skip-gram with negative sampling (SGNS)

Idea: 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*"?

→ take the learned classifier weights as the word embeddings

Big idea: self-supervision

- A word c that occurs near *apricot* in the corpus as the gold "correct answer" for supervised learning
- No need for human labels

Predicting if word *c* is a "neighbor"

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

Skip-gram training

Assume a +/- 2 word window, given training sentence:

…lemon, a [tablespoon of apricot jam,   a] pinch…

c1 c2 c3 c4

Goal: train a classifier that is given a candidate (word, context) pair

(apricot, jam)

(apricot, aardvark)

…

Assigns each pair a **probability**:

P(+|w, c): *c* is in the context of word *w*

P(−|w, c) = 1 − P(+|w, c)

Computing probability

One context word:

$$P(+|w,c) = \sigma(c \cdot w) = \frac{1}{1+\exp(-c \cdot w)}$$

$$\begin{aligned} P(-|w,c) &= 1 - P(+|w,c) \\ &= \sigma(-c \cdot w) = \frac{1}{1+\exp(c \cdot w)} \end{aligned}$$

Multiple context words:

$$\begin{aligned} P(+|w,c_{1:L}) &= \prod_{i=1}^{L} \sigma(c_i \cdot w) \\ \log P(+|w,c_{1:L}) &= \sum_{i=1}^{L} \log \sigma(c_i \cdot w) \end{aligned}$$

Example

…lemon, a [tablespoon of apricot jam,   a] pinch…

c1 c2 c3 c4

positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

negative examples -

t	c	t	c
apricot	aardvark	apricot	seven
apricot	my	apricot	forever
apricot	where	apricot	dear
apricot	coaxial	apricot	if

For each positive example we'll take *k* negative examples (here, *k*=2)

Learn the vectors

- Given the set of positive and negative training instances, and an initial set of embedding vectors
- The **goal of learning** is to adjust those word vectors such that we:
- **Maximize** the similarity of the target word, context word pairs $(w\ ,\ c_{pos})$ drawn from the **positive data**
- **Minimize** the similarity of the $(w\ ,\ c_{neg})$ pairs drawn from the **negative data**

Loss function

Maximize the similarity of the target with the actual context words, and minimize the similarity of the target with the *k* negative sampled non-neighbor words.

$$
\begin{aligned}
L_{\mathrm{CE}} &= -\log\left[P(+|w,c_{pos})\prod_{i=1}^{k}P(-|w,c_{neg_i})\right] \\
&= -\left[\log P(+|w,c_{pos})+\sum_{i=1}^{k}\log P(-|w,c_{neg_i})\right] \\
&= -\left[\log P(+|w,c_{pos})+\sum_{i=1}^{k}\log\left(1-P(+|w,c_{neg_i})\right)\right] \\
&= -\left[\log\sigma(c_{pos}\cdot w)+\sum_{i=1}^{k}\log\sigma(-c_{neg_i}\cdot w)\right]
\end{aligned}
$$

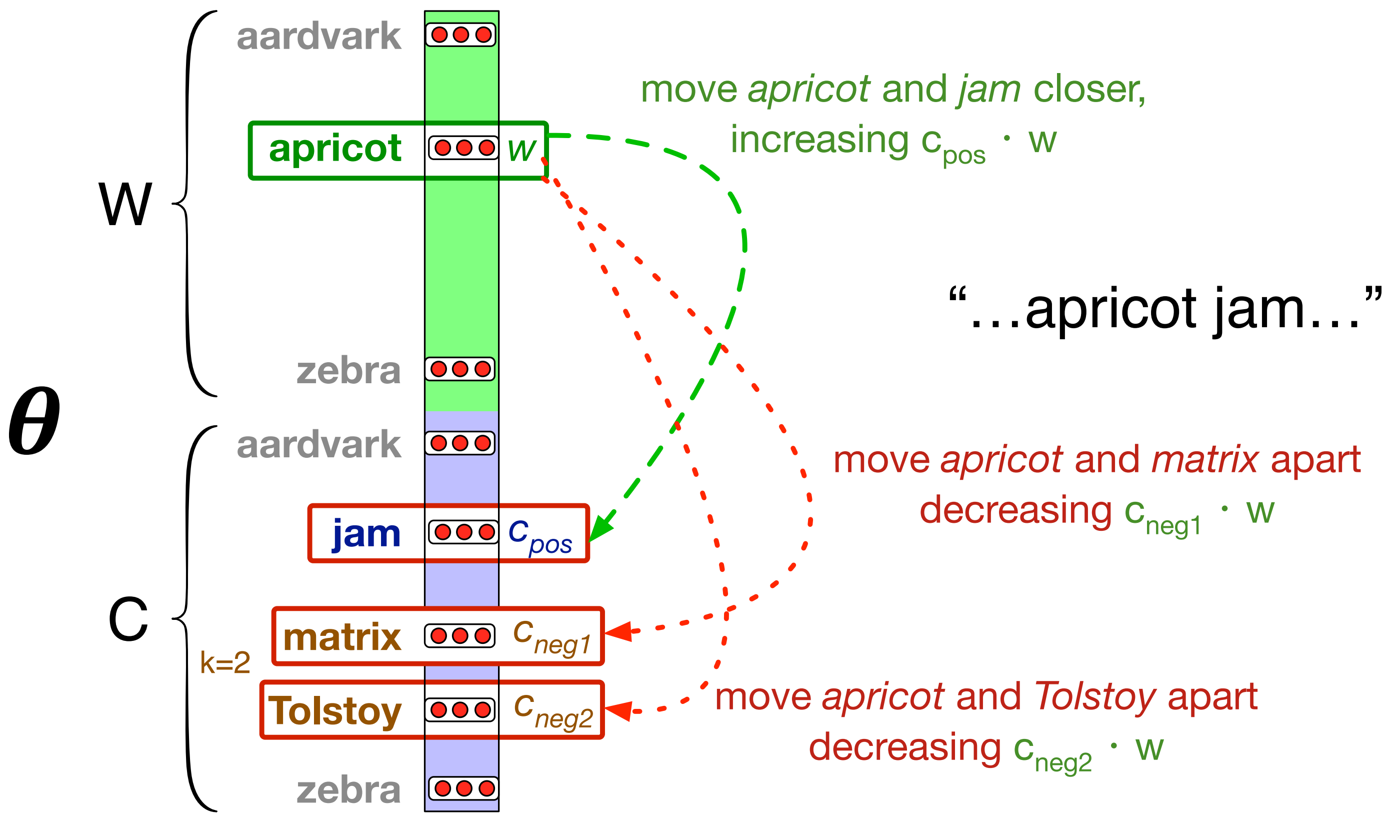
Learning the classifier

How to learn?

Gradient descent!

We'll adjust the word weights to

- make the positive pairs more likely
- and the negative pairs less likely,
- over the entire training set.



The derivatives of the loss function

$$L_{\mathrm{CE}} = -\left[\log \sigma(c_{pos} \cdot w) + \sum_{i=1}^{k} \log \sigma(-c_{neg_i} \cdot w)\right]$$

$$\frac{\partial L_{CE}}{\partial c_{pos}} = [\sigma(c_{pos} \cdot w) - 1]w$$

$$\frac{\partial L_{CE}}{\partial c_{neg}} = [\sigma(c_{neg} \cdot w)]w$$

$$\frac{\partial L_{CE}}{\partial w} = [\sigma(c_{pos} \cdot w) - 1]c_{pos} + \sum_{i=1}^{k} [\sigma(c_{\mathrm{neg}_i} \cdot w)]c_{neg_i}$$

Update weights

Start with randomly initialized C and W matrices, then incrementally do updates

$$
\begin{aligned}
c_{pos}^{t+1} &= c_{pos}^{t} - \eta[\sigma(c_{pos}^{t} \cdot w^{t}) - 1]w^{t} \\
c_{neg}^{t+1} &= c_{neg}^{t} - \eta[\sigma(c_{neg}^{t} \cdot w^{t})]w^{t} \\
w^{t+1} &= w^{t} - \eta\left[[\sigma(c_{pos} \cdot w^{t}) - 1]c_{pos} + \sum_{i=1}^{k}[\sigma(c_{\mathrm{neg}_i} \cdot w^{t})]c_{neg_i}\right]
\end{aligned}
$$

Get the embeddings

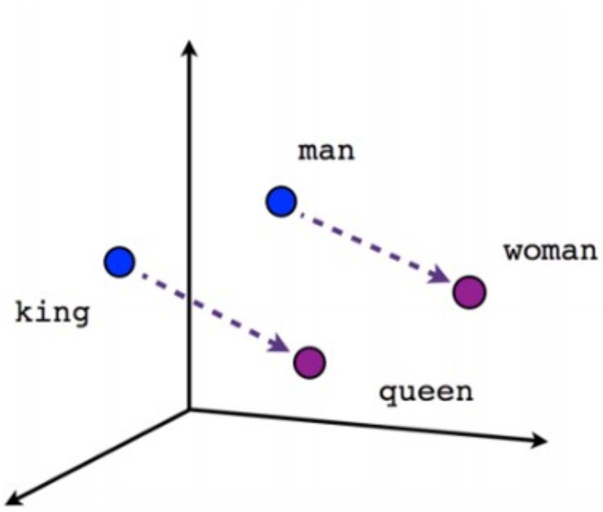
Skip-gram learns two sets of embeddings

Target embeddings matrix W

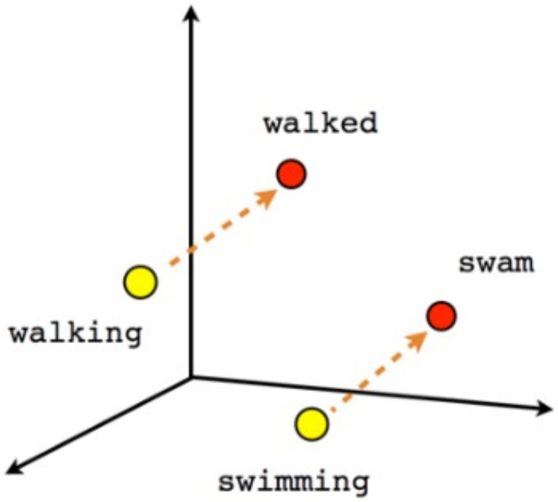
Context embedding matrix C

It's common to just add them together, representing word *i* as the vector $w_i + c_i$

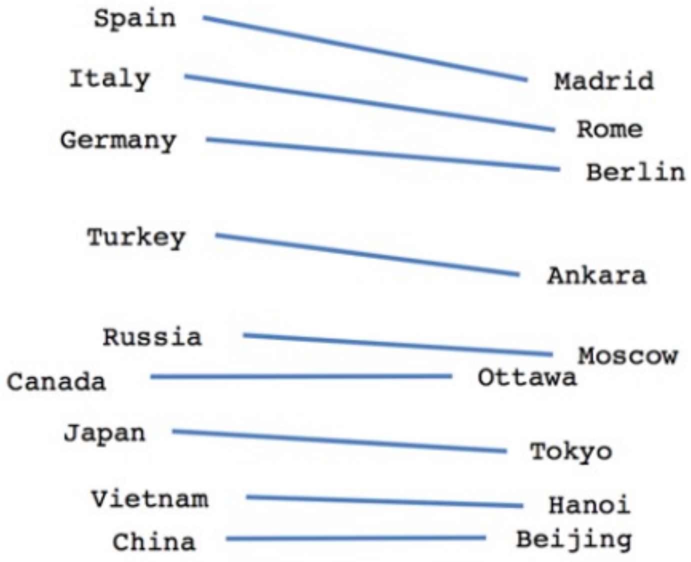
Evaluating word embeddings



Male-Female



Verb tense



Country-Capital

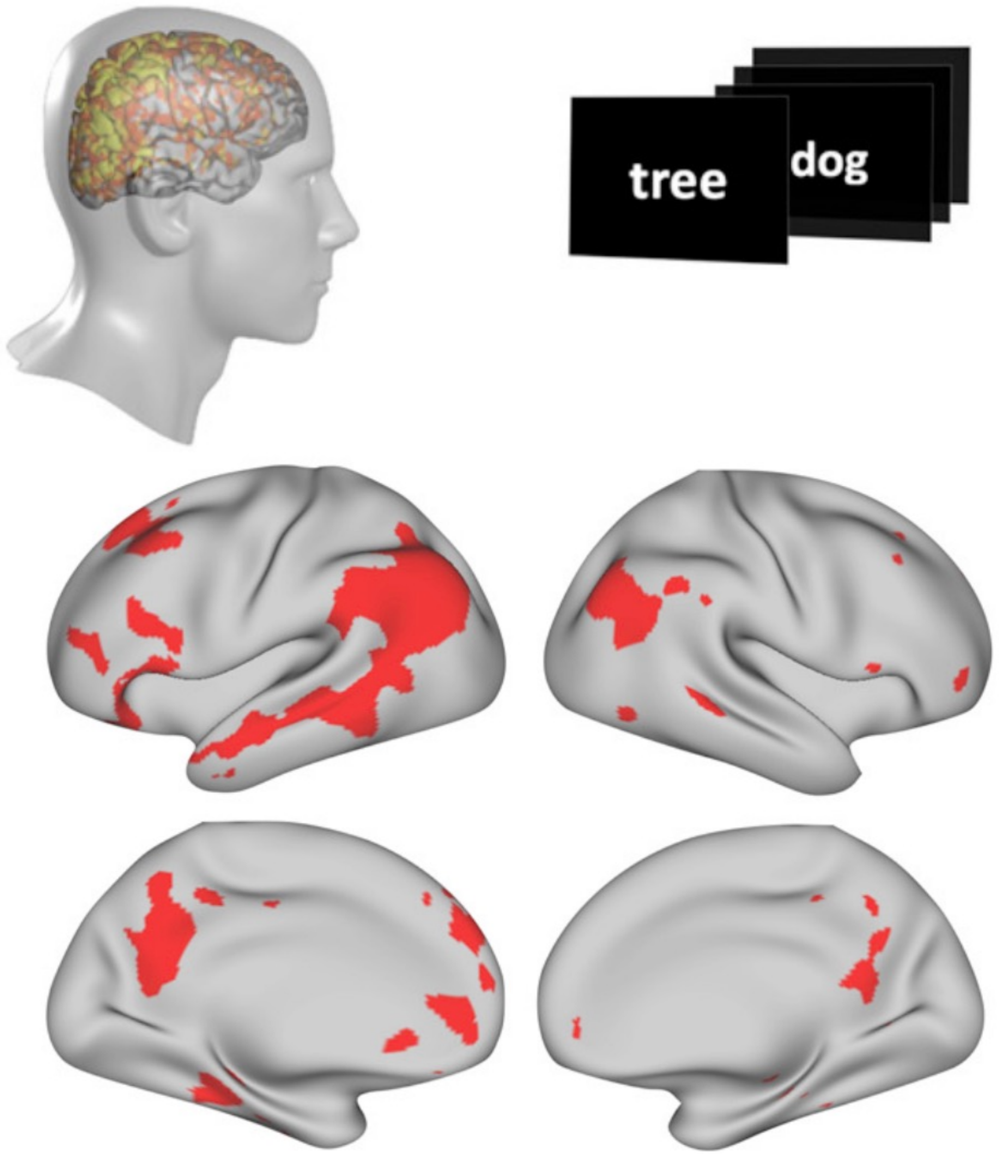
Against human judgement

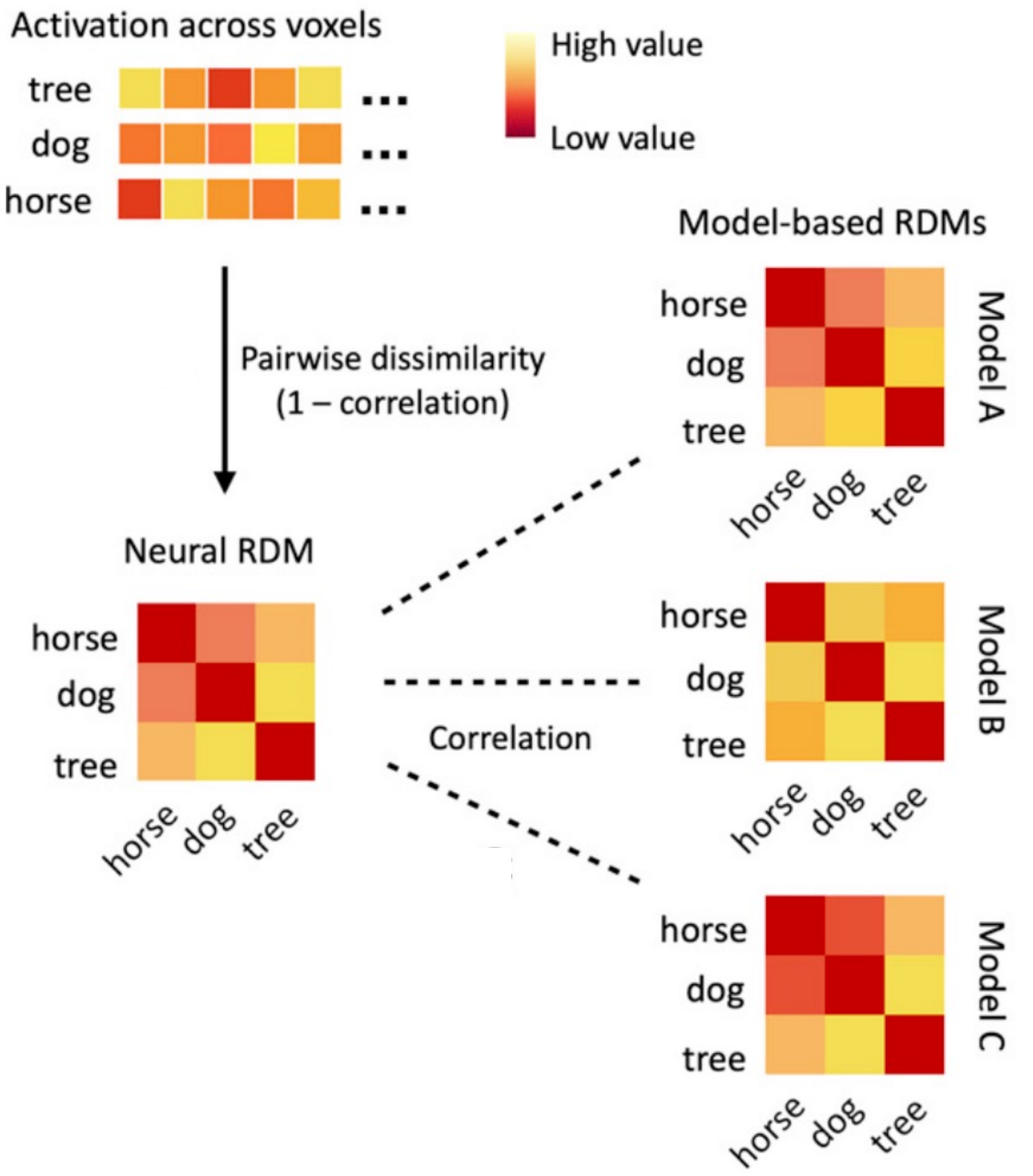
SimLex-999: Human rating on the similarity between 1000 pairs of words (scale: 0-10)

word1	word2	similarity
vanish	disappear	9.8
behave	obey	7.3
belief	impression	5.95
muscle	bone	3.65
modest	flexible	0.98
hole	agreement	0.3

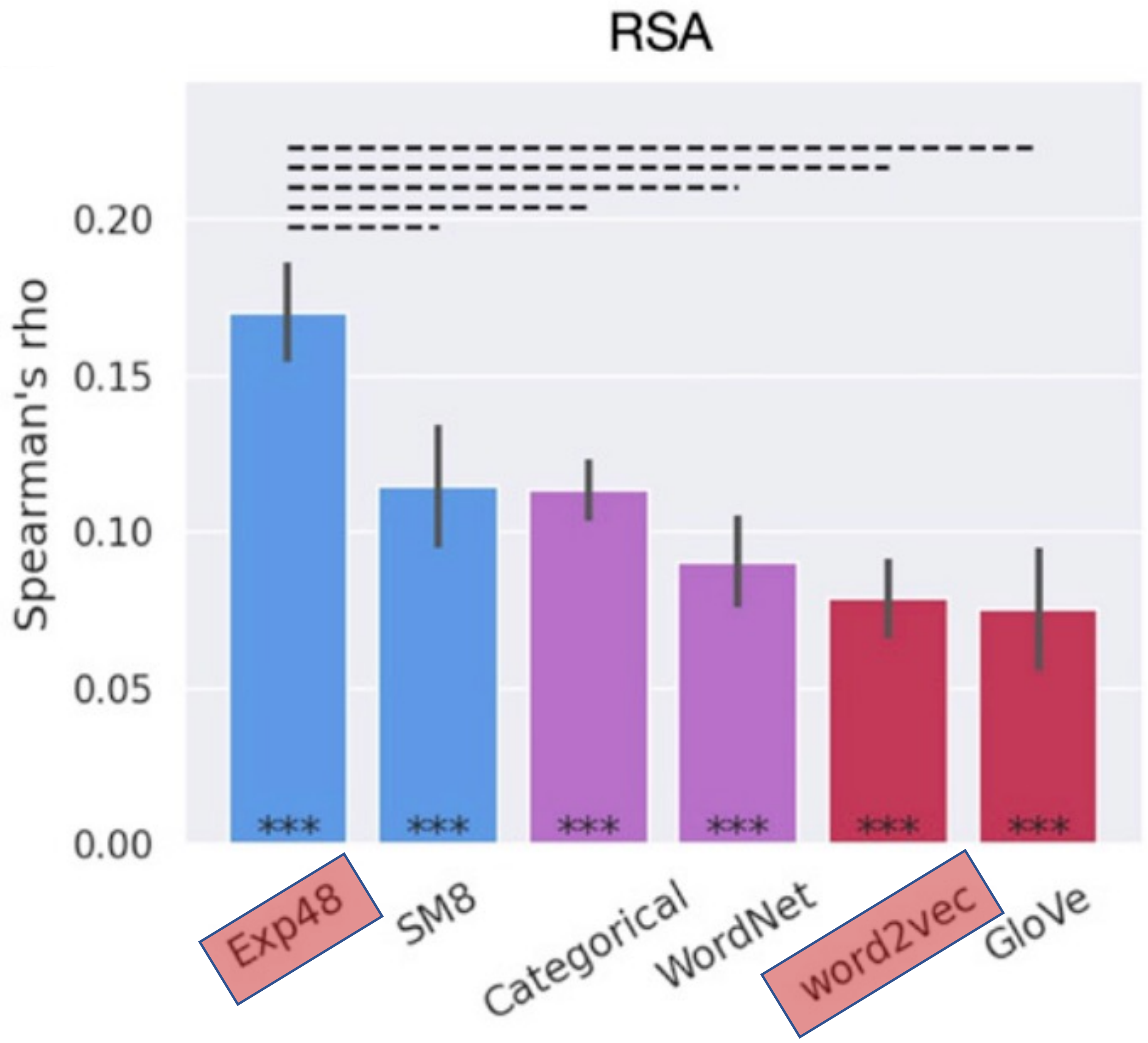
Calculate the correlation between the cosines of the word embeddings and the simlex-999 values

Against human brain data?





Against human brain data?



Word	Vision	Bright	Dark	Color	Pattern	Large	Small
ant	3.5484	0.3548	3.5806	3.9355	1.9355	0.0968	5.871
bicycle	5.3	1.1667	0.6333	1	2.1667	1.7	1.2667
farm	5.7097	1.1935	0.5161	1.7419	1.8065	5.0645	0.129
farmer	4.1786	0.5	0.3214	0.4286	0.6071	1.4286	0.6786
green	4.2963	1.7778	1	5.9259	1.5926	0.1852	0.1111
red	5	3.2857	1.25	6	1.4643	0.1071	0.0357
rocket	5.5	2.9333	0.7333	1.8667	1.9	5.6	0.2333
trust	0.3793	0.1379	0.0345	0.3103	0.2069	0.3103	0.069

To do

- Optional reading: **SLP** Ch6
- Do HW8